

AD-A060 670

AIR FORCE WEAPONS LAB KIRTLAND AFB N MEX  
FOURTH GENERATION COMPUTER ARCHITECTURES.(U)  
JAN 78 G W BREZINA

F/G 9/2

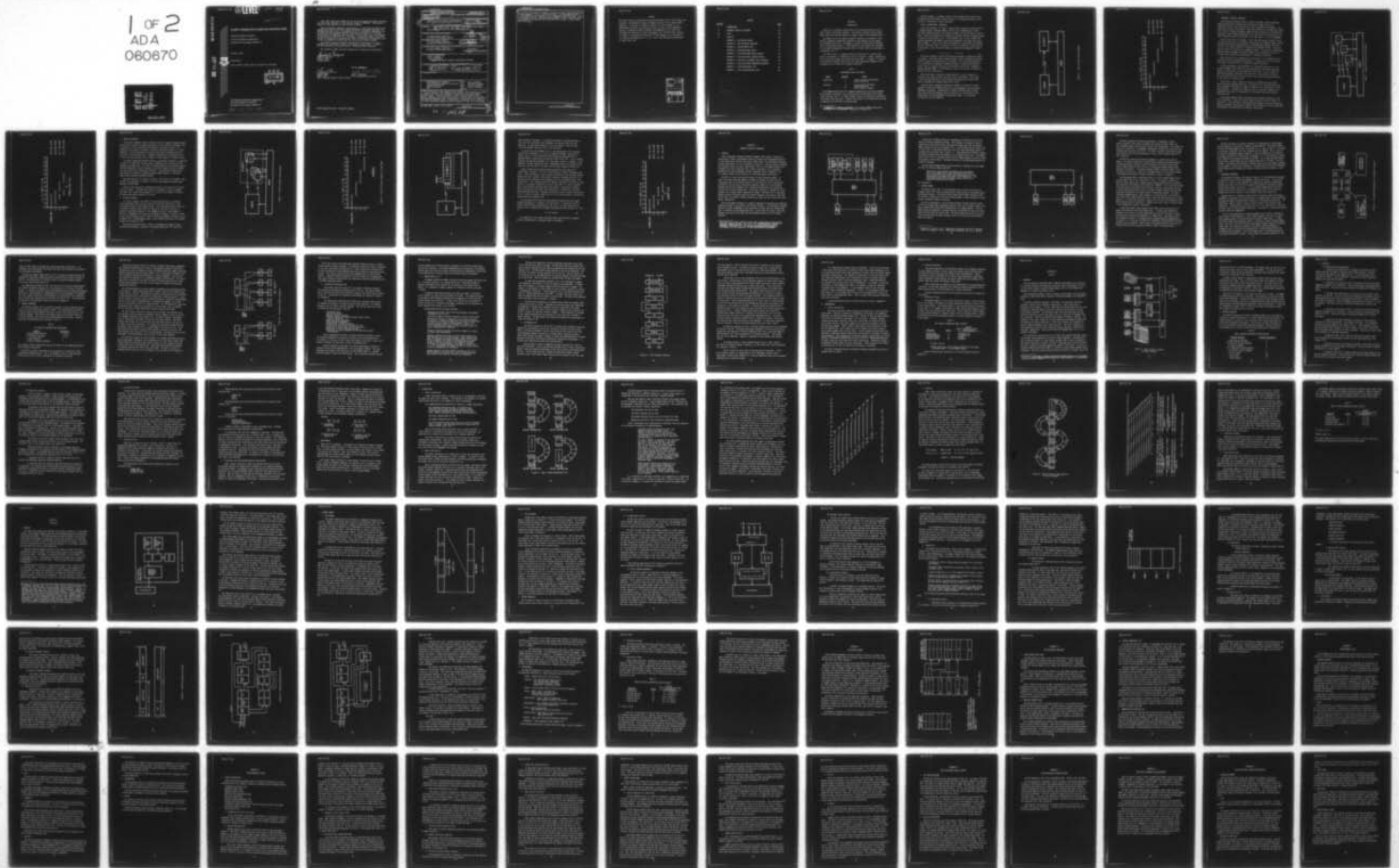
UNCLASSIFIED

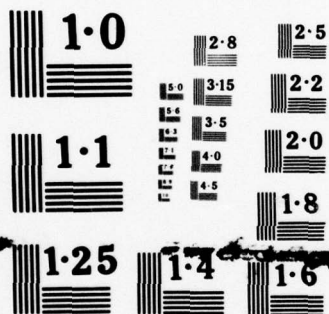
AFWL-TR-77-190

SBIE-AD-E200 129

NL

1 OF 2  
ADA  
060670





NATIONAL BUREAU OF STANDARDS  
MICROCOPY RESOLUTION TEST CHART

**2 LEVEL**

DPC

ADE200129

AD A060670

DDC FILE COPY

**FOURTH GENERATION COMPUTER ARCHITECTURES**

Captain Gerald W. Brezina

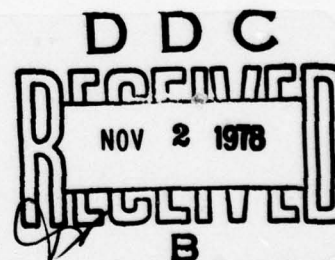
Air Force Weapons Laboratory

Kirtland Air Force Base, NM 87117

January 1978

Final Report

Approved for public release; distribution unlimited.



AIR FORCE WEAPONS LABORATORY  
Air Force Systems Command  
Kirtland Air Force Base, NM 87117

78 10 19 005



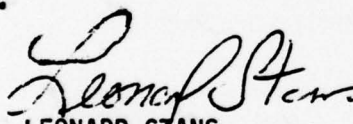
This final report was prepared by the Air Force Weapons Laboratory, Kirtland, Air Force Base, New Mexico. The Job Order Number is 99930000. Capt Gerald W. Brezina is the Laboratory Project Officer-in-Charge.

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

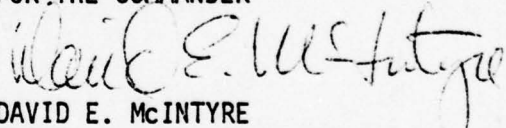
This report has been reviewed by the Office of Information (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

  
GERALD W. BREZINA  
Captain, USAF  
Project Officer

  
LEONARD STANS  
Major, USAF  
Chief, Advanced Systems Project Officer

FOR THE COMMANDER

  
DAVID E. MCINTYRE  
Chief, Computational Division

DO NOT RETURN THIS COPY. RETAIN OR DESTROY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM															
1. REPORT NUMBER <b>(14) AFWL-TR-77-190</b>	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER															
4. TITLE (and Subtitle) <b>(6) FOURTH GENERATION COMPUTER ARCHITECTURES</b>	5. TYPE OF REPORT & PERIOD COVERED <b>(9) Final Report</b>																
7. AUTHOR(s) <b>(10) Gerald W. Brezina</b> Captain Brezina		8. CONTRACT OR GRANT NUMBER(s)															
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Weapons Laboratory Kirtland Air Force Base, NM 87117		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS JON: <b>(16) 99930000</b> <b>(17) PE 62601F</b>															
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Weapons Laboratory Kirtland Air Force Base, NM 87117		12. REPORT DATE <b>(11) January 1978</b>															
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Air Force Weapons Laboratory Kirtland Air Force Base, NM 87117		13. NUMBER OF PAGES 104															
16. DISTRIBUTION STATEMENT (of this Report) <b>(12) 108p.</b> Approved for public release; distribution unlimited.		15. SECURITY CLASS. (of this report) UNCLASSIFIED															
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) <b>(18) SBIE</b> <b>(19) AD-E200 129</b>																	
18. SUPPLEMENTARY NOTES																	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) <table border="0"> <tr> <td>Fourth-generation computers</td> <td>TI-ASC</td> <td>Serial Processor</td> </tr> <tr> <td>Fourth-generation computer architecture</td> <td>CRAY</td> <td>Concurrent Processor</td> </tr> <tr> <td>Computer architecture</td> <td>CRAY-1</td> <td>Parallel Processor</td> </tr> <tr> <td>Vector processors</td> <td>STAR</td> <td>Pipeline Processor</td> </tr> <tr> <td>ASC</td> <td>STAR-100</td> <td></td> </tr> </table>			Fourth-generation computers	TI-ASC	Serial Processor	Fourth-generation computer architecture	CRAY	Concurrent Processor	Computer architecture	CRAY-1	Parallel Processor	Vector processors	STAR	Pipeline Processor	ASC	STAR-100	
Fourth-generation computers	TI-ASC	Serial Processor															
Fourth-generation computer architecture	CRAY	Concurrent Processor															
Computer architecture	CRAY-1	Parallel Processor															
Vector processors	STAR	Pipeline Processor															
ASC	STAR-100																
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) <p>The architecture of the computer central processor holds the key to increasing speed of operation after electronic transmission lines rates become a limiting factor. The serial processor was the first and simplest type of central processor, and although many of today's machines appear to have their own unique architectures, their processors are, in most cases, only combinations of three architectural types: (1) concurrent; (2) parallel; and (3) pipeline. The Texas Instruments Advanced Scientific Computer (TI-ASC) can be configured</p>																	

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 55 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

78 10 19 005  
013 150

1/18

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

to form up to four parallel arithmetic pipelines with each pipeline constructed of eight segments. It has a cycle time of 80 ns. The Cray Research CRAY-1 is a concurrent pipeline computer with 12 independent functional units. It has a cycle time of 12.5 ns. The Control Data STAR-100 is also a concurrent pipeline computer which operates upon string and arrays of data. It has a cycle time of 40 ns. *K*

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

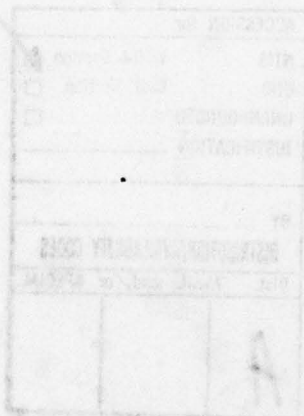
## PREFACE

The author wishes to acknowledge the expert assistance of Mrs. Shirley Wells who patiently and methodically typed the numerous drafts of this report without complaint. I also want to express my thanks to Lt Karl Kortkamp for his proofreading the drafts for technical accuracy and for his many suggestions. Section II, Advanced Scientific Computer and Appendixes A, B, C and D are reprinted with permission of Texas Instruments, Inc. and AFIPS Press; Section III, CRAY-1 is reprinted with permission of CRAY Research, Inc.; and Section IV, STAR-100 and Appendixes E, F, G, H, I and J are reprinted with permission of Control Data Corp.

ACCESSION for		
NTIS	White Section	<input checked="" type="checkbox"/>
DDC	Buff Section	<input type="checkbox"/>
UNANNOUNCED		<input type="checkbox"/>
JUSTIFICATION _____		
BY _____		
DISTRIBUTION/AVAILABILITY CODES		
Dist.	AVAIL.	and/or SPECIAL
A		

## CONTENTS

<u>Section</u>		<u>Page</u>
I	INTRODUCTION	3
II	ADVANCED SCIENTIFIC COMPUTER	16
III	CRAY-1	33
IV	STAR-100	51
	APPENDIX A - ASC/CENTRAL MEMORY	73
	APPENDIX B - ASC/PERIPHERAL PROCESSOR	75
	APPENDIX C - ASC/ARITHMETIC UNIT	78
	APPENDIX D - ASC/INSTRUCTION TIMING	81
	APPENDIX E - STAR-100/STORAGE ACCESS CONTROL	88
	APPENDIX F - STAR-100/CENTRAL PROCESSOR MODES	89
	APPENDIX G - STAR-100/I/O ASSEMBLY AND DISASSEMBLY	90
	APPENDIX H - STAR-100/VIRTUAL ADDRESSING MECHANISM	91
	APPENDIX I - STAR-100/REGISTER FILE	96
	APPENDIX J - STAR-100/INSTRUCTION TYPES	100



## SECTION I

## INTRODUCTION

Common to all computer architectures are the three operational units of input, output, and central processor. The central processor can be further broken down into the three subunits of memory, control, and arithmetic/logic.

The memory unit of the central processor resembles an electronic filing cabinet, completely indexed and accessible to the computer via the control and arithmetic/logic units. The memory is generally constructed of magnetic core or semiconductor devices. These devices are partitioned into thousands of individually addressable locations (words or bytes) that can store data or instructions (ref. 1).

The control unit of a computer executes instructions by a process that occurs in two phases: (1) Recognition, and (2) Execution. Table 1 sequentially lists the specific events occurring within each phase. In general, the control unit fetches and decodes instructions and sends appropriate commands to the arithmetic unit to direct the arithmetic operation.

Table 1

## INSTRUCTION PHASES AND EVENTS

<u>Phase</u>	<u>Sequence</u>	<u>Event</u>
Recognition	1	Fetch instruction from memory
	2	Decode instruction
Execution	3	Fetch operand from memory
	4	Perform operation
	5	Store operand to memory

The arithmetic/logic unit of a computer contains the circuitry to perform the arithmetic operations, such as addition and multiplication, and to make logic decisions; that is, to choose between several alternatives or conditions and perform prescribed tasks. Normally the selection is accomplished by making a comparison and performing a program branch.

1. Fundamentals of Computer Programming, Air Training Command Study Guide, SG 30BR5141-1-2, 29 September 1972, pp 2-55 through 2-75.

Working together, the memory, control, and arithmetic/logic units can be constructed to perform their various functions in a sequential manner to form a serial or conventional central processor.

## 1. SERIAL (CONVENTIONAL) PROCESSOR

The serial processor in figure 1 shows the basic three-unit construction of a computer central processor; memory, control, and arithmetic/logic units. The figure shows the control unit access to memory for the fetching of instructions. After the instruction is decoded, the control directive (lightning arrow) is passed to the arithmetic unit. The arithmetic unit, in turn, fetches the necessary operands from memory, performs the operation, and stores the result back to memory. Since in this type of architecture, instructions are executed serially, the total time,  $T$  to execute  $n$  similar instructions is just the sum of the times,  $t_i$  each instruction takes to accomplish the sequence in table 1. That is,

$$T = t_1 + t_2 + \dots + t_n = nt \quad (1)$$

The serial processor instruction execution procedure is graphically shown in figure 2. The figure shows a sample program of four instructions and their times of execution plotted against a time scale. For illustrative purposes, all instructions are assumed to complete execution in three time periods or cycles, one for recognition and two for execution; and the "R#" represents a machine register number. In the example, a total of 12 cycles is required to execute the four instructions.

Note from figure 2 that the ultimate speed of a serial processor is limited by the extent to which the cycle time can be reduced. This places a limit upon the execution of user programs and essentially restricts the types of mathematical problems that can be solved.

State-of-the-art computer architectures attempt to overcome the serial computer limitation to produce results at a rate necessary to solve current mathematical problems. Although many of today's machines appear to have their own unique architectures, their processors are, in most cases, only combinations of three fundamental architectural types: (1) Concurrent, (2) Parallel, and (3) Pipeline.

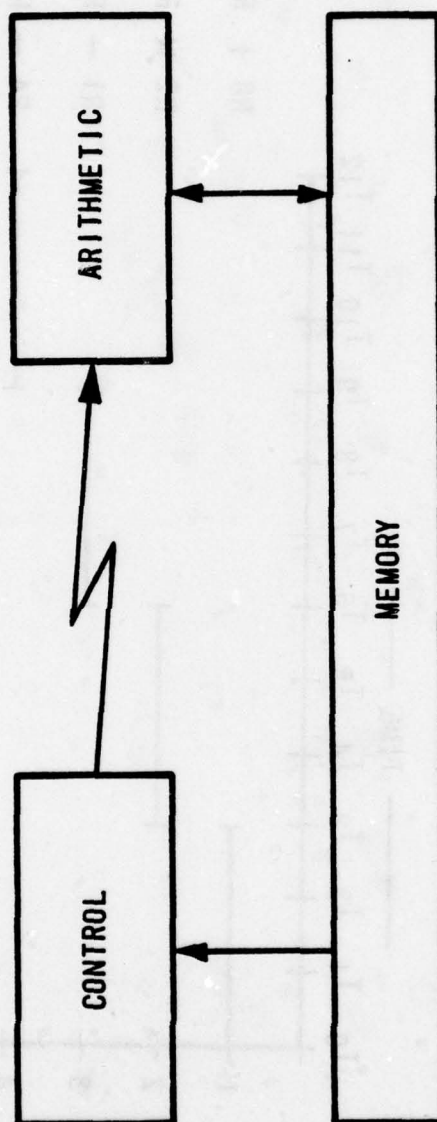


Figure 1. Serial Processor Block Diagram

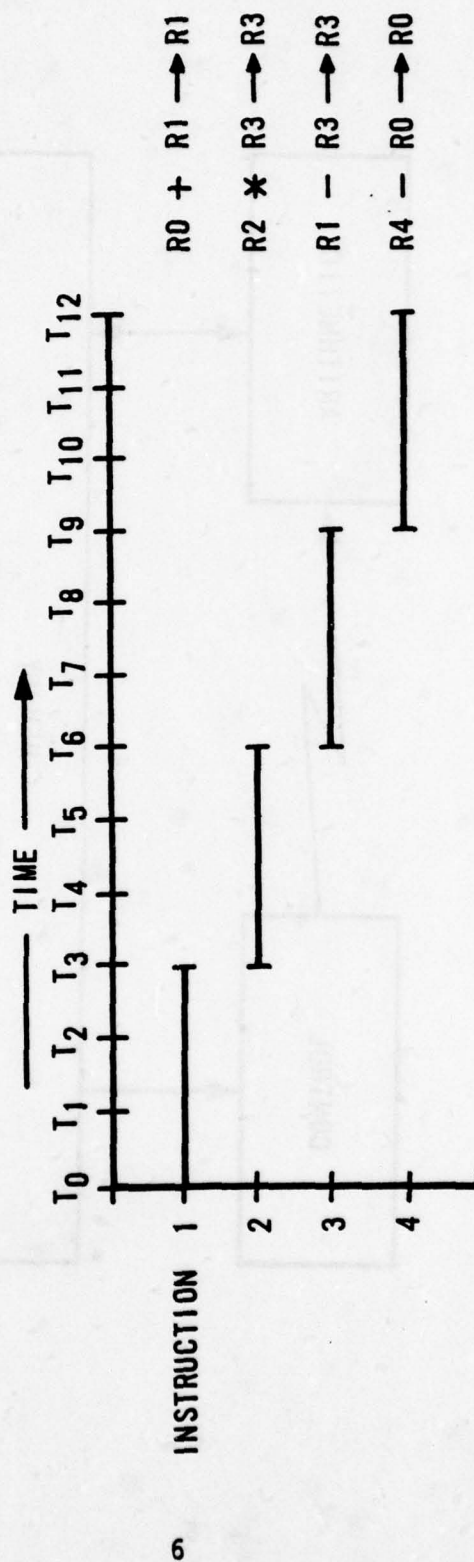


Figure 2. Serial Processor Sequence of Operations

## 2. CONCURRENT (OVERLAP) PROCESSOR

The concurrent processor shown in figure 3 achieves a greater computing speed than the serial processor by dividing the arithmetic unit into separate functional units and permitting them to operate simultaneously. The functional units are constructed so that each performs only one operation; therefore, several different arithmetic operations may be going on at the same time. The heart of this hardware mechanism lies in the ability of the control unit to detect operations which do not conflict with each other, i.e., are independent.

Conflicts may occur in two ways. First, the most recently fetched instruction may use the result of a preceding instruction as an operand; hence, instruction execution must be postponed until the operand-result becomes available for use. This type of conflict is labeled as an operand conflict. The second type of conflict is labeled as an operator conflict. This conflict materializes when the recently fetched instruction needs to use a functional unit that is currently busy completing the execution of a preceding instruction.

When the control unit interprets an instruction, it will determine when no conflicts arise with preceding instructions. If there are no conflicts, then the present instruction will be allowed to complete execution.

To take advantage of the simultaneity of instruction execution in a concurrent processor, the control unit can be modified to begin the recognition phase of an instruction at the beginning of each cycle. Figure 4 shows the concurrent processor execution of the same set of instructions as figure 2. Due to the modification of the control unit, each instruction begins execution at the start of a processor cycle.

Instructions 1 and 2, contain no operator or operand conflicts; hence, each instruction completes execution. Instruction 3, on the other hand, uses the R3 operand of instruction 2; hence, it must wait for instruction 2 to complete execution. Instruction 4, does not use an instruction 3 operand, but nevertheless must wait for instruction 3 to relinquish use of the subtract functional unit.

In the above example, eight cycles were used to complete execution of the four instructions. Clearly, the total time to execute a set of  $n$  similar instructions is less than that of the conventional computer, since the recognition phase and nonconflicting execution phases of each instruction can be accomplished simultaneously with both phases of previous instructions.

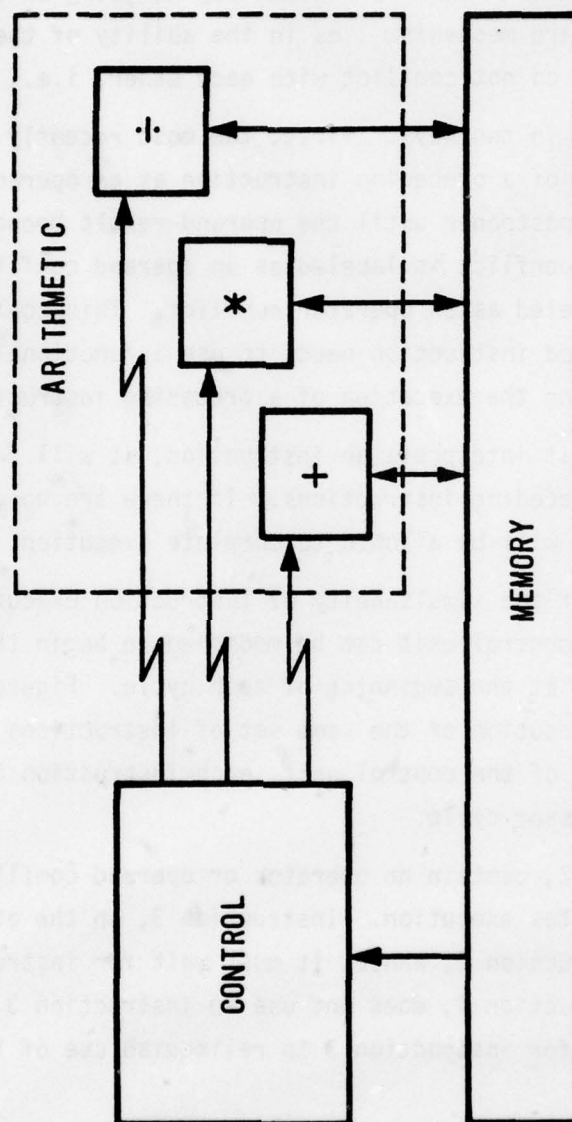


Figure 3. Concurrent Processor Block Diagram

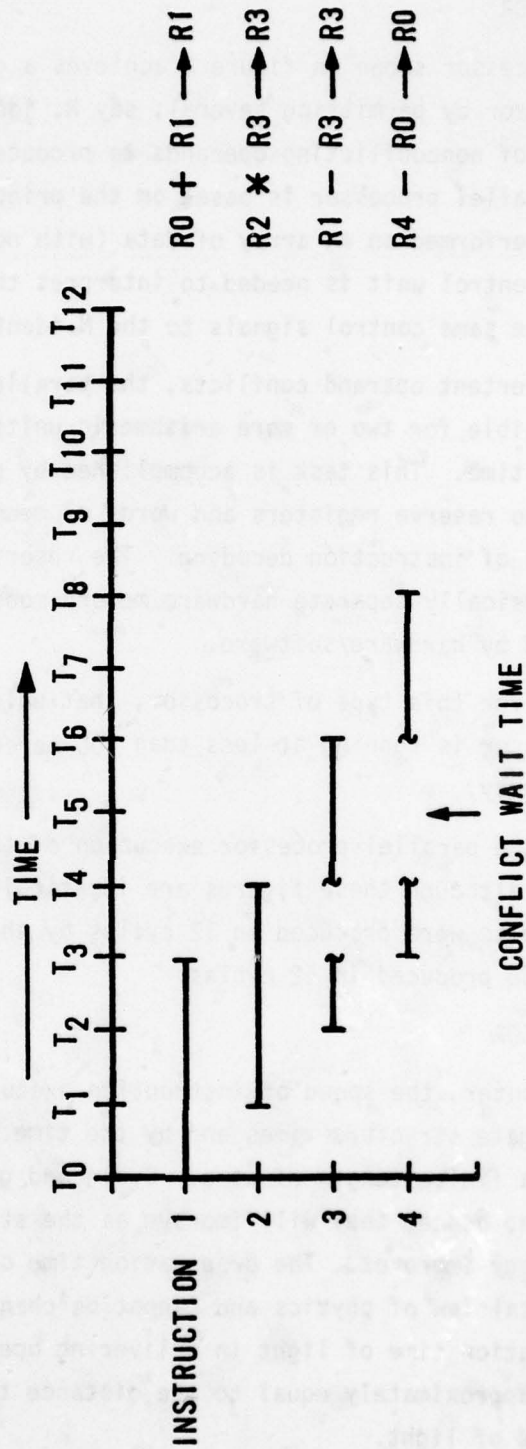


Figure 4. Concurrent Processor Sequence of Operations

### 3. PARALLEL PROCESSOR

The parallel processor shown in figure 5 achieves a greater computing speed than a serial processor by permitting several, say  $N$ , identical arithmetic units to operate on a set of nonconflicting operands to produce  $N$  results. The operation of the parallel processor is based on the principle that if a single operation is to be performed on an array of data (with no operand conflicts) then only a single control unit is needed to interpret the instruction, and it, in turn, can pass the same control signals to the  $N$  identical arithmetic units.

To prevent inadvertent operand conflicts, the parallel processor is organized so that it is impossible for two or more arithmetic units to change the same operand at the same time. This task is accomplished by providing some hardware/software mechanism to reserve registers and words of memory for each arithmetic unit upon initiation of instruction decoding. The reservation mechanism may take the form of physically separate hardware memory modules or contiguous memory being divided by hardware/software.

One should note for this type of processor, that unless all arithmetic units are full, the processor is running at less than 100 percent utilization thus, reducing its efficiency.

Figure 6 shows the parallel processor execution of the same set of instructions as figure 2. Although these figures are identical, one must remember that where four results were produced in 12 cycles by the example in figure 2,  $4N$  results may now be produced in 12 cycles.

### 4. PIPELINE PROCESSOR

In a serial computer, the speed of instruction execution is limited by cycle and hardware gate structure times and by the time required for light to propagate along a finite length of wire. Cycle and gate structure times are a function of design; hence, they will improve as the state of the art in engineering technology improves. The propagation time of light, however, is fixed by a fundamental law of physics and cannot be changed. The delay caused by the fixed propagation time of light in delivering operands from memory to the arithmetic unit is approximately equal to the distance traveled by the operands divided by the speed of light.

The pipeline processor shown in figure 7 overcomes this speed of light limitation by accessing a second set of operands before the first result has

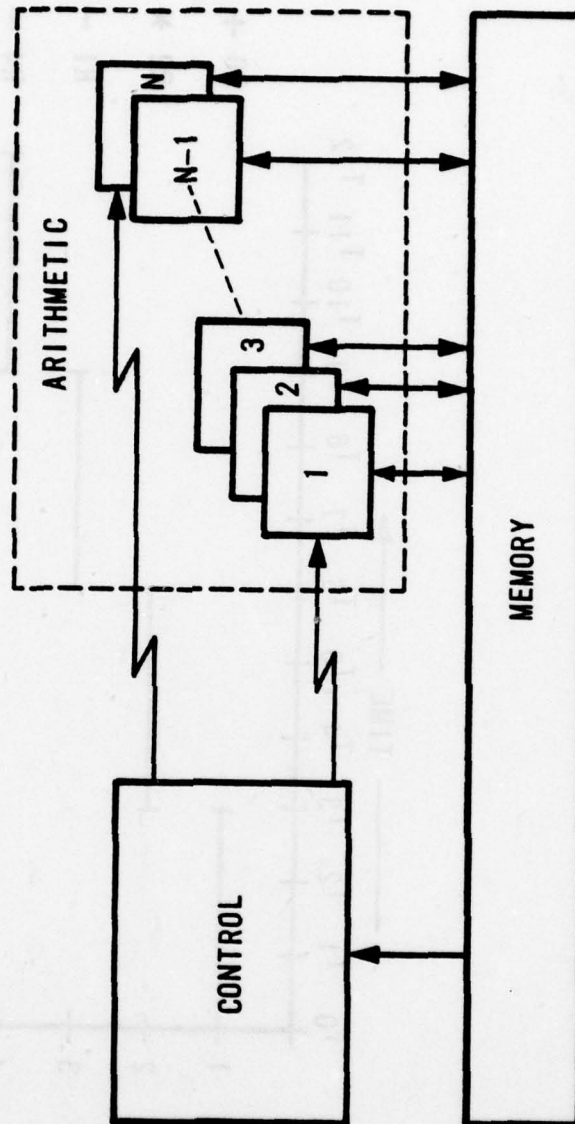


Figure 5. Parallel Processor Block Diagram

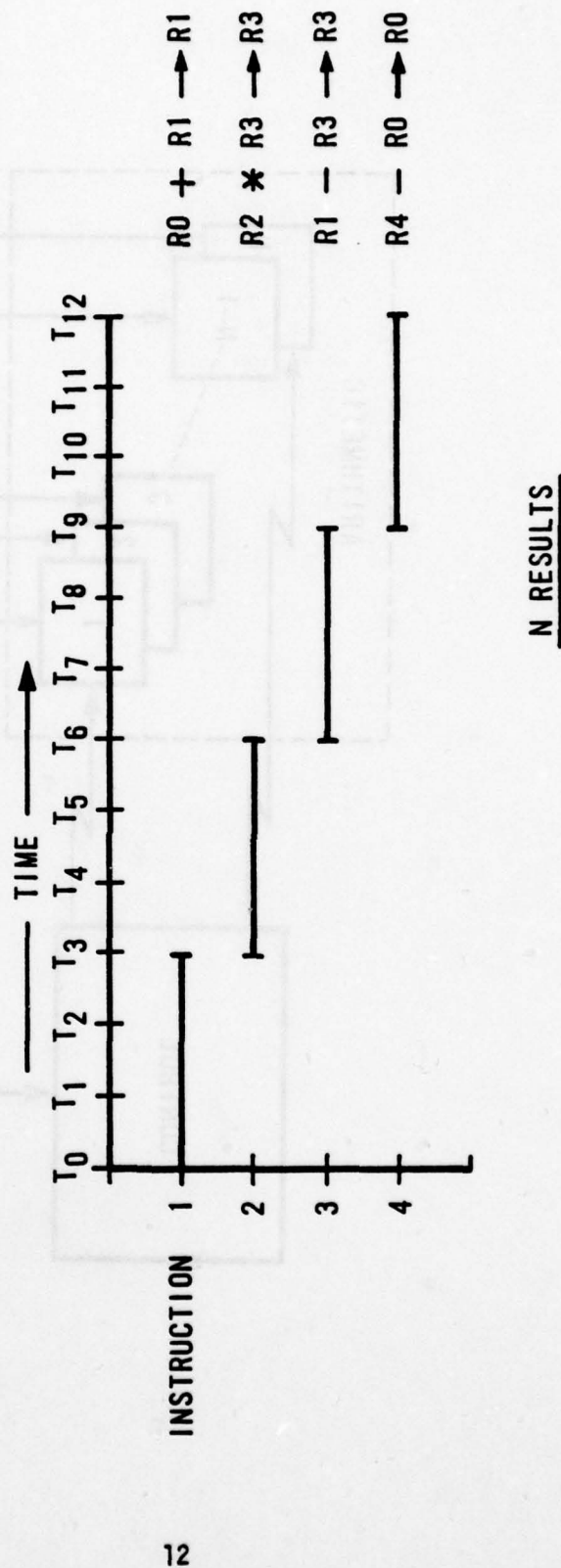


Figure 6. Parallel Processor Sequence of Operations

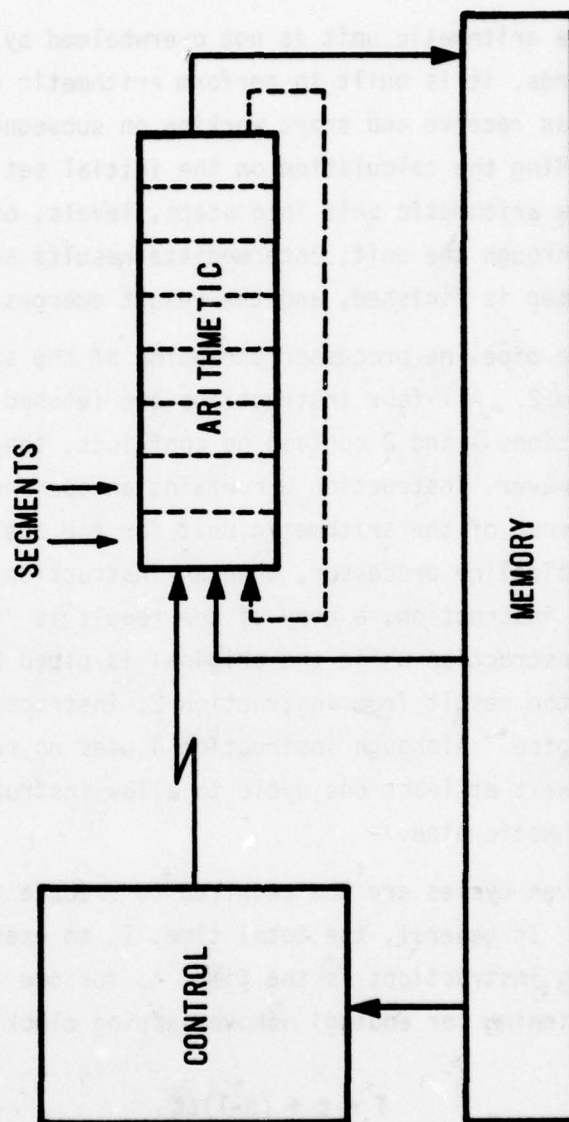


Figure 7. Pipeline Processor Block Diagram

been returned to the memory. As accessing continues, the operands begin to stack up at the entrance to the arithmetic unit and wait their turn for execution. As the pipeline begins to fill, the speed of light no longer limits the apparent execution rate of the processor.

To insure that the arithmetic unit is not overwhelmed by the increased availability of operands, it is built to perform arithmetic operations in stages; that is, it can receive and start working on subsequent sets of operands before finishing the calculation on the initial set. This is accomplished by dividing the arithmetic unit into steps, levels, or segments. As an operation is passed through the unit, intermediate results are passed to each step until the last step is finished, and the result emerges from the unit.

Figure 8 shows the pipeline processor execution of the same set of instructions of figure 2. All four instructions are fetched from memory every cycle. Since instructions 1 and 2 contain no conflicts, they continue passing through the pipe. However, instruction 3 contains an operand conflict, so it must wait at the entrance of the arithmetic unit for the operand-result from instruction 2. In a pipeline processor, when an instruction is waiting for the result of a preceding instruction, a copy of the result is "short circuited" back to the waiting instruction while the original is piped back to memory. Once it has received the result from instruction 2, instruction 3 may then continue through the pipe. Although instruction 4 uses no results from instruction 3, it must also wait at least one cycle to allow instruction 3 to begin its journey down the arithmetic pipe.

In the example seven cycles are now required to execute the four sample program instructions. In general, the total time,  $T$ , to execute a set of  $n$  similar nonconflicting instructions is the time,  $t$ , for one instruction to execute, plus  $n-1$  beginning (or ending) nonoverlapping clock periods,  $\Delta t$ . That is,

$$T = t + (n-1)\Delta t \quad (2)$$

The remainder of this report describes three state-of-the-art computers which use combinations of the above types of processors.

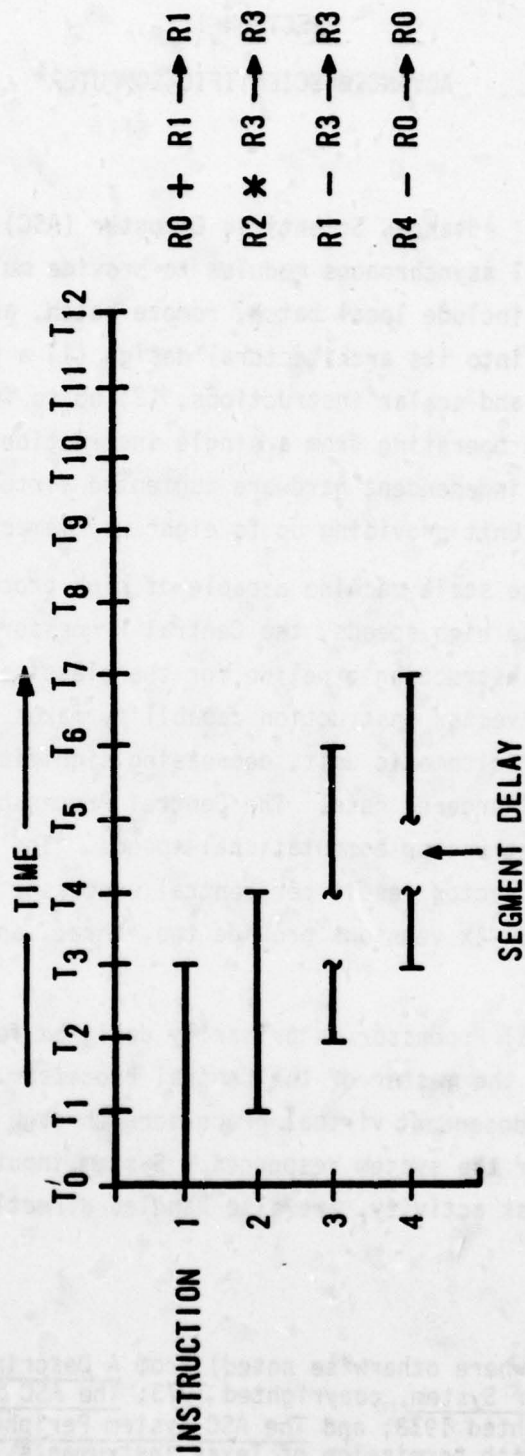


Figure 8. Pipeline Processor Sequence of Operations

## SECTION II

### ADVANCED SCIENTIFIC COMPUTER\*

#### 1. OVERVIEW

Texas Instruments' Advanced Scientific Computer (ASC) shown in figure 9 is constructed of several asynchronous modules to provide multiprogrammed operation on job streams which include local batch, remote batch, and interactive terminals. The ASC incorporates into its architectural design (1) a pipeline Central Processor with separate vector and scalar instructions, (2) up to four Memory Buffer Unit--Arithmetic Unit pairs operating from a single instruction, (3) a Peripheral Processor with eight independent hardware augmented Virtual Processors, and (4) a Memory Control Unit providing up to eight way memory interleaving.

The ASC is a large scale machine capable of high processing speeds. In order to achieve these high speeds, the Central Processor utilizes an 80 ns clock period and an instruction pipeline for the classical scalar instructions. In addition, the ASC vector instruction capability makes use of the pipeline technique within the arithmetic unit, decreasing significantly the processing time for sets of well-ordered data. The Central Processor is available in three versions for different vector computational speeds. The basic ASC 1x (one-pipe) machine provides one vector result per central processor clock period, while the ASC-2x, ASC-3x and ASC-4x versions provide two, three, and four results per clock period, respectively.

The ASC Peripheral Processor is primarily designed for executing the operating system and serves as the master of the Central Processor. The Peripheral Processor consists of eight independent virtual processors through which the computational load is scheduled for the system resources. System input and output, other than magnetic tape and disk activity, are also handled directly by the Peripheral Processor.

---

\* Reprinted (except where otherwise noted) from A Description of the Advanced Scientific Computer System, copyrighted 1973; The ASC System - Central Processor, copyrighted 1973; and The ASC System Peripheral Processor, copyrighted 1974 with permission of Texas Instruments Incorporated.

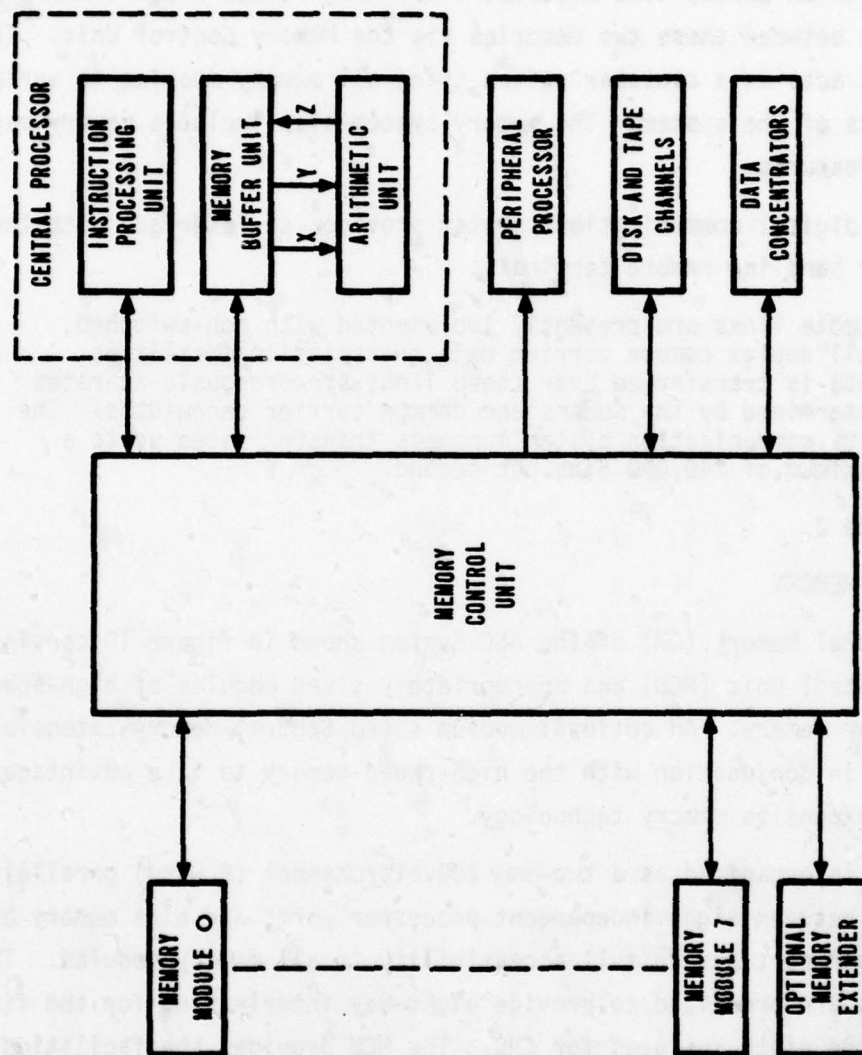


Figure 9. ASC System Configuration

The ASC Central Memory System can accommodate up to 18 million (M) 32-bit words of directly addressable storage. This storage can be made up from any combination of high speed bipolar semiconductor memory and lower performance memory through the use of nine memory ports. The main memory has an access time of 140 ns and is interleaved eight ways, while the lower performance memory is nonleaved with an access time of about 1  $\mu$ s. High speed block transfers, however, are possible between these two memories via the Memory Control Unit. The Memory Control Unit acts as a crossbar switch tying all memory modules to various processing elements of the system. The memory system also includes memory mapping and protection features.

The ASC digital communications system provides an interface with the common carriers for handling remote terminals.

Remote links are presently implemented with non-switched, full duplex common carrier data transmission facilities. Data is transferred over these links synchronously at rates determined by the modems and common carrier bandwidths. The data communication system supports transfer rates up to a maximum of 240,000 bits per second.

See reference 2.

## 2. CENTRAL MEMORY

The Central Memory (CM) of the ASC System shown in figure 10 consists of a Memory Control Unit (MCU) and appropriately sized modules of high-speed bipolar semiconductor memory. An optional medium speed Central Memory Extension (CME) can be used in conjunction with the high-speed memory to take advantage of slower and more inexpensive memory technology.

The MCU is organized as a two-way 256-bit/channel (8 word) parallel access traffic net between eight independent processor ports and nine memory buses, with each processor port having full accessibility to all memory modules. The nine memory buses are organized to provide eight-way interleaving for the first eight buses with the ninth bus used for CME. The MCU provides the facilities for controlling access from the eight processor ports to a CM having a 24-bit address space (16 M words).

- 
2. Watson, W.J. and Carr, H.M., "Operational Experiences with the TI Advanced Scientific Computer," AFIPS - Conference Proceedings, Vol 43, pp. 389-397.

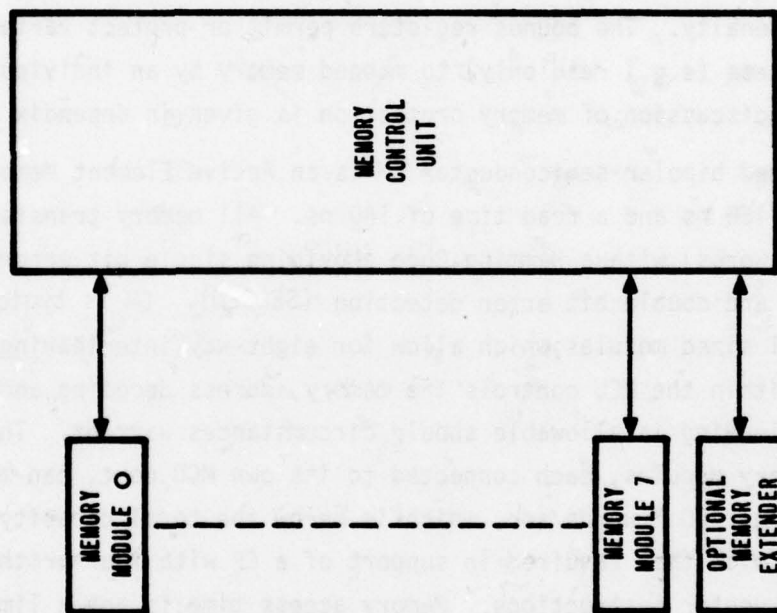


Figure 10. ASC Central Memory

The MCU is designed to operate asynchronously, independent of cable delays, processor clock rates, and memory unit access and cycle times, and will therefore support multiple data transfer rate requirements. This capability allows for flexibility to accommodate improvements in memory and processor technologies which develop in the future. The MCU is capable of handling a maximum data transfer rate of 80 M words/sec/port giving a total transfer capacity of 640 M words/sec.

The MCU also contains the necessary hardware for controlling access to memory. This memory protection is accomplished by mapping and bounds registers. The mapping registers prevent different programs from interfering with each other's memory areas and allow the use of discontinuous memory pages without execution time penalty. The bounds registers permit or protect various classifications of access (e.g., read only) to mapped memory by an individual program. A more detailed discussion of memory protection is given in appendix A.

The high-speed bipolar semiconductor CM is an Active Element Memory having a cycle time of 160 ns and a read time of 140 ns. All memory transfers are 256 bits (eight words) with a Hamming Code providing single bit error detection and correction, and double bit error detection (SECDED). CM is typically divided into eight equal sized modules which allow for eight-way interleaving; however, a patch board within the MCU controls the memory address decoding and less than eight-way interleaving is allowable should circumstances warrant. The eight interleaved memory modules, each connected to its own MCU port, can maintain a total data rate of 400 M words/sec, which is below the total capacity of the MCU; however, it is twice that required in support of a CP with four arithmetic units when processing vector instructions. Memory access time is not a limiting factor in the computational speed of the machine.

In some ASC systems, more processors and control units require access to memory than there are individual memory ports. In these cases, Memory Port Expanders are utilized to provide additional ports and are utilized to service the devices not requiring the bandwidth of a memory port. Each Memory Access Port Expander provides a 1:4 expansion with a maximum bandwidth degradation of 10 percent, i.e., from rate of a 80M 32-bit words/sec to approximately 72M 32-bit words/sec with an Expander. Memory Access Port Expanders may be treed upon one another with priorities resolved on either a fixed or distributed basis.

The CME, available as an option, provides for large amounts of medium-speed low-cost memory to be utilized in support of high-speed CM. This type of memory is an Active Element Memory with a cycle time of 1  $\mu$ s. and, like CM, it is accessed in eight word increments, utilizing a Hamming Code for SECDED. CME is a part of the directly addressable memory; therefore, it may be addressed by any processor or channel controller for instructions or operands. It is also possible to effect block transfers of information between CM and CME. This is made possible because CME has both a normal memory bus and a Memory Access Port to the MCU. The block transfer, initiated by the Peripheral Processor, specifies the two memory addresses and the number of words to be transferred. The CME transfers the data autonomously at 40 M words/sec and informs the Peripheral Processor when the transfer is complete.

### 3. PERIPHERAL PROCESSOR

The Peripheral Processor (PP) shown in figure 11 is a multiprocessor designed to perform the control and data management functions of the ASC. Its principle function is to serve as the processor for executing the operating system which provides the control facility for the entire ASC system--all peripheral devices, direct access storage, and the CP. It also provides communication with input/output (I/O) devices, functions as the system monitor, and fulfills those job requests which do not require a rich arithmetic instruction repertoire.

Because the PP is intended to perform control functions rather than execute mathematical algorithms, the instruction set is oriented toward control operations and does not require multiplication, division, or floating point operations. The instruction format is similar to that of the CP using a 32-bit word for each instruction. Instructions are provided for bit (1 bit), byte (8 bits), half word (16 bits), and full word (32 bits) operations. The typical PP instruction requires two 85 ns cycles for completion.

The PP is time shared by up to eight programs, each of which is executed by one of eight Virtual Processors. Each Virtual Processor (VP) acts as a stand-alone computer having its own instruction and control capabilities. A Single Word Buffer (SWB) with associated control is utilized to allow overlapped memory access among VPs. Each VP also has its own program counter along with arithmetic, index, base, and instruction registers, and shares a Read Only Memory, an Arithmetic Unit, an Instruction Processing Unit, a Central Memory Buffer and 64 Communications Registers. Use of the common units is distributed among the VPs using 16 single 85 ns cycles. When an equally distributed sequence of time

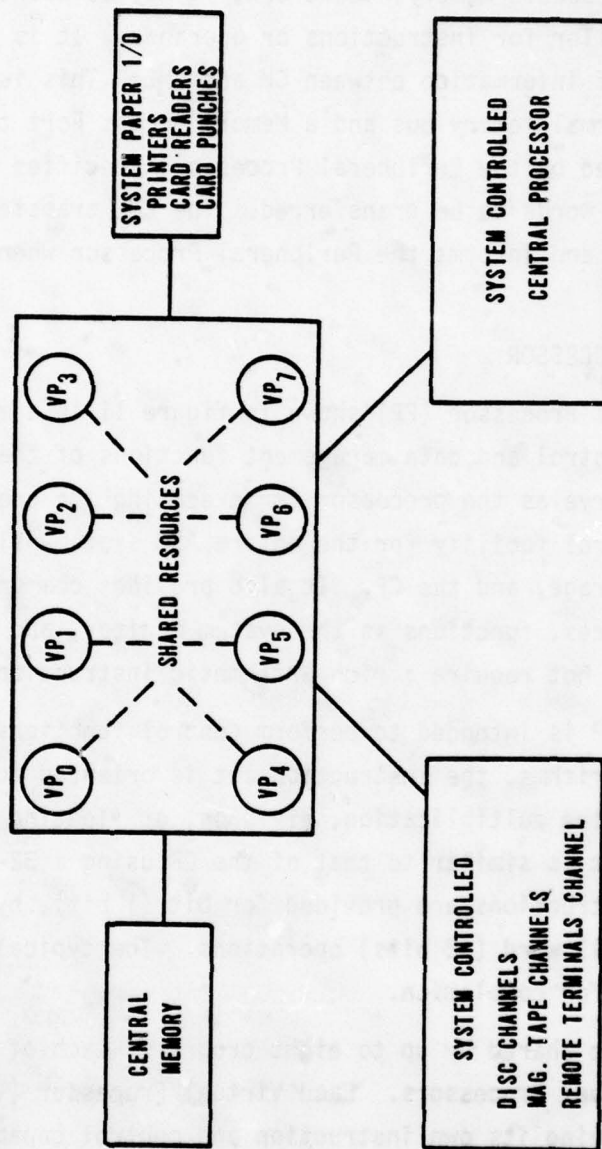


Figure 11. ASC Logical Organization of Peripheral Processor

units are used, each of the eight VPs receives two 85ns cycles every 1.4  $\mu$ s. otherwise, the distribution of available time units can be varied to suit particular processing requirements.

The Read Only Memory (ROM) within the PP is utilized for program storage and execution of those short operating system routines which are highly utilized by the VPs. The ROM consists of 4K 32-bit words of nonvolatile memory elements with a cycle time of less than 85 ns.

The Communications Register (CR) file serves as the principal storage medium for control information necessary for the coordination of all parts of the ASC System. It contains sixty-four 32-bit word registers which are program addressable by the VPs. Synchronization of communications is achieved between all processors (CP, VPs, channel controllers, and peripheral unit controllers) from interpretation of status bits received from all devices into the CR file. A more detailed description of the SWB, ROM, VP and CR is given in appendix B.

#### 4. CENTRAL PROCESSOR

The Central Processor (CP) is primarily designed to execute program instructions in a single instruction pipeline stream. This stream contains a mixture of 32-bit scalar (single operand) and vector (array) instructions with 16-, 32-, or 64-bit operands. The CP has an adjustable primary clock period (cycle) of 80 ns and contains forty eight 32-bit program-addressable registers. Register functions and designations are indicated in table 2.

Table 2

#### ASC REGISTER FUNCTIONS AND DESIGNATIONS

<u>Function</u>	<u>Designation</u>
16 Base Address Registers	A and B
16 Arithmetic Registers	C and D
8 Index Registers	I
8 Vector Parameter Registers	V

The V group is used to extend the instruction format for the complete specification of vector instructions.

The pipeline concept, described in the Introduction, is applied by Texas Instruments throughout the CP. The CP is designed so that a single execution unit can have up to 12 scalar instructions in progress at one time.

The basic structure of the CPs in figure 12 contains three major components: The Instruction Processing Unit (IPU) for nonarithmetic stages of instruction processing for the CP instruction stream; the Memory Buffer Unit (MBU) to provide operand interfacing with the Central Memory; and a corresponding Arithmetic Unit (AU) to perform the specified arithmetic or logical operations. Each two and four pipeline CP consists of a corresponding number of MBU-AU pairs. The figure shows that a multipipe CP provides for parallel execution below the IPU; thus, the ASC takes advantage of not one, but two types of processor architectures; pipeline and parallel.

Up to 36 instructions in various stages of execution can be overlapped within a four pipe CP. Eight instructions can be located in each of four pipes and one in each level of the IPU. This means that there are 20 positions for instructions in the two pipe CP and 12 positions for instructions in the one pipe CP. In the IPU any mixture of scalar or vector instructions may be in execution simultaneously since, in any of the CP configurations, the interaction between an IPU, MBU, and AU is equivalent to that of a single pipeline. The IPU performs all decisions pertaining to the routing of instructions to various pipes; thus, any MBU-AU pair is not aware of other MBU-AU pairs that may comprise the system.

The flow of data in the CP is from the IPU to the MBU, from the MBU to the AU, and then from the AU back to the MBU for stores to memory or back to the IPU for arithmetic results to the register file. In order to ensure a continuous flow of instructions from memory, the IPU has two 8-word buffers. One buffer contains the current set of instructions while the other receives the next set of eight instructions from memory. All instruction processing, except that part performed by the arithmetic unit, is accomplished in the four-level IPU. Each MBU contains three pairs of 8-word buffer registers; one pair for each of two input operands and a third pair for an output operand. The buffers serve to allow overlapped, interleaved memory access for each of the vector operands.

One significant feature of the CP hardware is an operand look-ahead capability which causes memory references to be requested prior to the time of actual need. Double buffering in multiple 8-word (octet) buffers for each pipeline provides a smooth data flow to and from each AU. The eight-level pipelined AU achieves its highest sustained flow rate in the vector mode. Typically, a result is produced every 80 ns per AU, or an average of 20 ns per result for an ASC-4X CP.

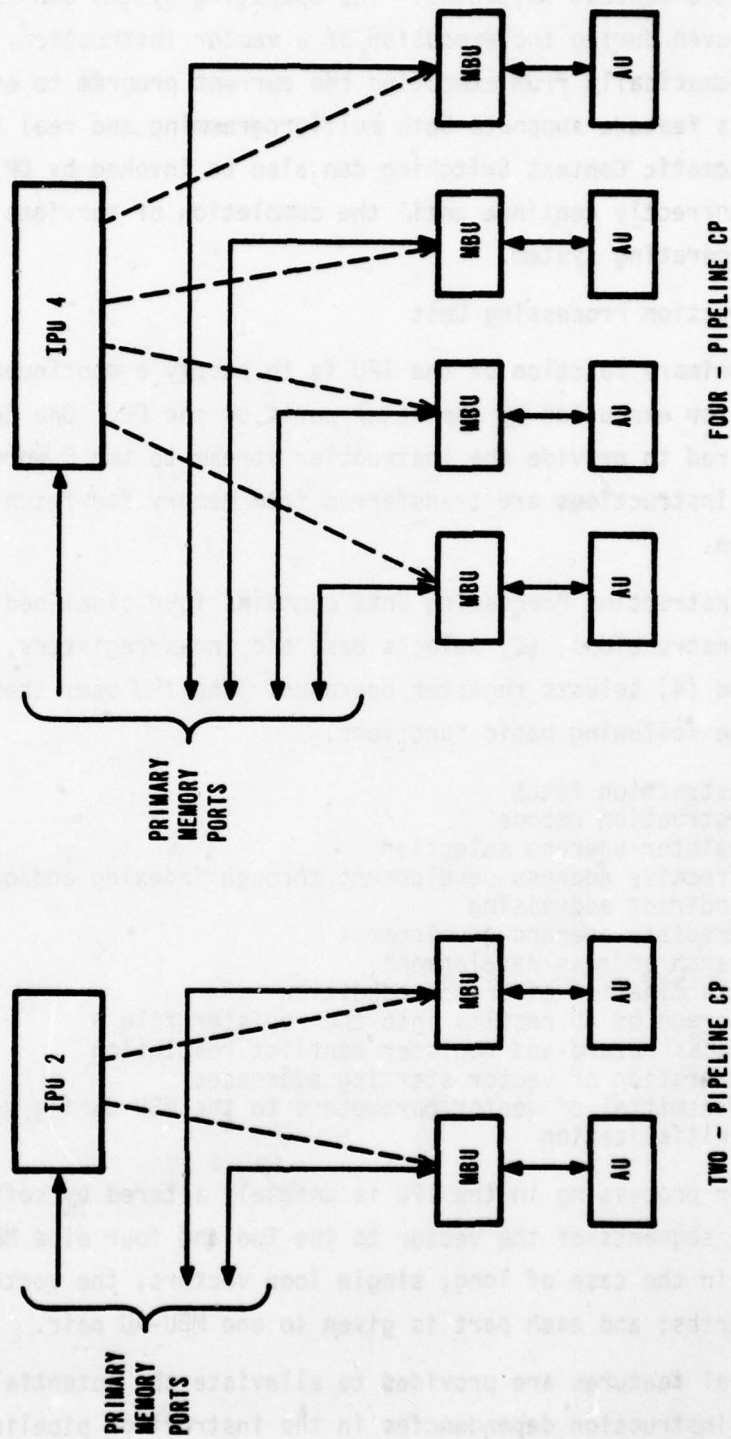


Figure 12. ASC Central Processor Structure

The CP can be time sliced among many programs through the use of a feature called Automatic Context Switching. The operating system can invoke an interrupt at any time, even during the execution of a vector instruction, causing the CP to switch automatically from executing the current program to executing the next program. This feature supports both multiprogramming and real time process control. Automatic Context Switching can also be invoked by CP programs when they cannot correctly continue until the completion of services performed by the PP resident operating system.

a. Instruction Processing Unit

The primary function of the IPU is to supply a continuous stream of instructions for execution by the other parts of the CP. One Central Memory port is required to provide the instruction stream to two 8-word (octet) buffers in the IPU. Instructions are transferred from memory for fetching or storing of information.

The Instruction Processing Unit contains four pipelined levels which (1) decodes instructions, (2) selects base and index registers, (3) develops addresses, and (4) selects register operands. The IPU uses these four levels to perform the following basic functions.

- Instruction fetch
- Instruction decode
- Register operand selection
- Effective address development through indexing and/or indirect addressing
- Immediate operand development
- Branch address development
- Determination of branch condition
- Storage of AU results into the register file
- Scalar hazard and register conflict resolution
- Generation of vector starting addresses
- Transmittal of vector parameters to the MBU during vector initialization

Vector processing in the IPU is uniquely altered by software in order to distribute segments of the vector to the two and four pipe MBU-AU pairs. For example, in the case of long, single loop vectors, the vector is split into halves or fourths; and each part is given to one MBU-AU pair.

Several features are provided to alleviate the potential problems of branches and instruction dependencies in the instruction pipeline. The Load-Look-Ahead instruction, used extensively by the FORTRAN compiler, increases execution speed of closed loop instructions. This instruction provides the IPU

control hardware with advance address information to facilitate uninterrupted instruction processing. Instruction dependencies are recognized by the hardware. The IPU scans the instruction stream and distributes the independent instructions across MBU-AU pairs to insure proper, yet efficient execution sequences.

b. Memory Buffer Unit

The Memory Buffer Unit (MBU) provides an interface between the CM and the AU. Its primary function is to supply the AU with a continuous stream of operands from memory and to provide for the storing of the results back to memory. All references to memory, whether for fetching or storing, are made in octets.

The MBU has three double buffers, one octet per buffer, called the X and Y buffers for input and the Z buffers for output. This double buffering is provided so that pipeline processing can be sustained at a high rate with minimal memory access conflicts. In fact, the Z buffer can be transferred directly to the X or Y buffers so that memory references are not necessary for scalar operands which reside in the Z buffer.

The MBU performs the following functions:

Accepting the initial vector starting addresses and parameter information from the IPU.

Fetching the memory operands requested by scalar instructions.

Retention of 16 words in temporary X and Y buffer registers for possible "scratch pad" operations involving data contained in the two most recently referenced memory octets. This temporary storage capability increases by a factor of 1, 2, 3, or 4 depending upon whether an ASC-1x, ASC-2x, ASC-3x, or ASC-4x configuration is installed.

Storage of register operands into central memory as a result of scalar store instructions.

Temporary retention of 8 words in the Z-buffer register for data destined for one CM octet address. Data stored by this means is released to CM when the octet address of write data at the Arithmetic Unit output is different than the octet address of the data contained in the Z-buffer registers. This temporary storage capability increases by a factor of 1, 2, 3 or 4 depending upon whether an ASC-1x, ASC-2x, ASC-3x, or ASC-4x configuration is installed.

Update capability from the Z-buffer registers to the X or Y buffer registers for keeping the X and Y registers current when they are being used for "scratch pad" operations.

During scalar operations, effective addresses developed by the index unit in the IPU are routed to the MBU. The MBU then obtains one operand from the CM location specified by the effective address and one operand from the IPU Register File. The MBU presents these operands to the AU for processing. The AU places its results into the register file of the IPU. When AU results are to be stored into CM, the MBU receives the effective address into which the data are to be stored. After AU processing, the MBU stores the data into memory.

For the scalar operations, buffers X and Y are alternated for memory read operations, and buffer Z is used for memory write operations. In either case, the strategy is to invoke a memory cycle only when one is needed. For example, a read request for data within an octet currently residing in a buffer is terminated at the buffer. A write operation into a previously defined write octet is likewise terminated at the buffer. An actual read cycle occurs only when the required data are not within a current octet. An actual write operation occurs only when a new write octet is defined.

For most vector operations, two operand data strings are fetched, while a result data string is stored. Addresses for sustaining the vector operations are computed in the MBU using parameters initially specified by the vector parameter file. Buffers X and Y supply strings of numbers to be processed, and buffer Z accepts the resultant string of numbers. Triple buffering is provided so that vector processing can be sustained at a high rate with a minimum of memory limiting. The MBU supplies the consecutive operands to be processed and stores the results in CM.

#### c. Arithmetic Unit

The primary function of the CP AU is to perform the arithmetic operations specified by the operation code of the instruction currently at the AU level. There is one AU per pipeline in the CP, each having an 80 ns basic cycle time. A distinguishing feature of the AU is its eight section pipeline structure. These eight sections are shown in figure 13, which also demonstrates how different sections of the AU are utilized for execution of particular instructions, i.e., floating point addition and fixed point multiplication.

Four sections in figure 13 comprise the basic structure of a floating point add instruction. Each of the sections performs parts of other instructions; however, they are primarily partitioned in this way to improve the floating point add time. Each of these sections is capable of operating on double length operands so that vector double length instructions can proceed at the clock rate.

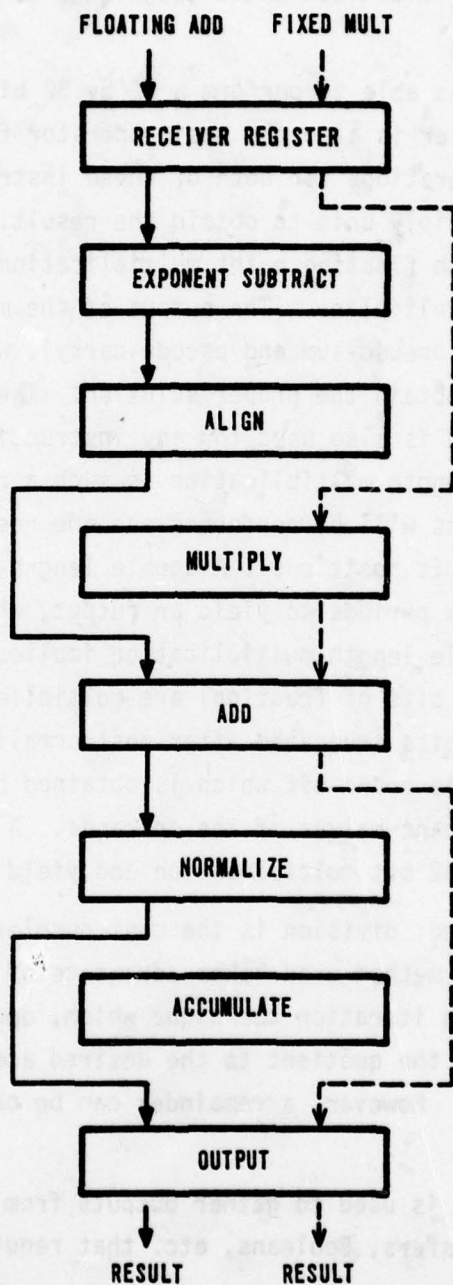


Figure 13. ASC Arithmetic Pipeline

The align section is used to perform right shifts in addition to the floating point alignment for add. The normalize section is used for all normalization requirements and will also perform left shifts for fixed point operands. The add section employs second level look-ahead techniques to perform both fixed and floating point additions.

The multiply unit is able to perform a 32 by 32 bit multiplication in one clock period. The multiplier is also the basic operator for the divide instruction, and double length operations for both of these instructions require several iterations through the multiply unit to obtain the result. Fixed point multiplications and single length floating point multiplications are available after only one pass through the multiplier. The output of the multiply unit is two words of 64 bits each (the pseudo-sum and pseudo-carry), which must be added in the accumulate section to obtain the proper solution. The accumulate section is similar to the add unit and is also used for any instruction which needs to form a running total. Double length multiplication is such a case, as three separate 32 by 32 bit multiplications will be performed and the results then added in the accumulator in the proper bit positions. A double length multiplication would therefore require six clock periods to yield an output, while single length would require only four. A double length multiplication implies that two 64-bit floating point numbers (56 bits of fraction) are multiplied to yield a 64-bit result with the low order bits truncated after postnormalization. This multiplication ignores a possible round-off which is obtained by making a fourth pass with the two least significant halves of the operands. A fixed point multiplication will perform a 32 by 32 bit multiplication and yield a 64-bit result.

As would be expected, division is the most complex operation to be performed by the AU in the ASC. The method used takes advantage of the fast multiplication capabilities and employs an iteration technique which, upon a specified number of multiplications, will form the quotient to the desired accuracy. This method does not form a remainder. However, a remainder can be obtained under program control.

The output section is used to gather outputs from all other sections and also to do simple transfers, Booleans, etc. that require only one clock period for execution in the AU.

The AU is used for both fixed and floating point instructions. Fixed point negative numbers are represented in two's complement notation, while hexadecimal floating point numbers are in sign and magnitude along with an exponent represented by an excess 64 number.

All floating point operands must be normalized. A guard digit consisting of four least significant bits is provided to avoid loss of one hexadecimal digit of accuracy which would result from truncation of results from prior double length additions or subtractions. The addition of these bits is sufficient because the only times that normalization with a possibility of loss of accuracy would be required involves a shift of only one hexadecimal digit. Normalized operands are required for the guard digit to be of maximum use. For example, in the case of addition, if the exponents are equal, no alignment is required; therefore, the guard digit is not necessary. If the exponents differ by one, the guard digit will retain significant information. Finally, if the exponents differ by more than one, it can be shown that the result to be normalized will require at most a shift of one hexadecimal digit. Thus, the guard digit contains information that can be retrieved.

A more detailed description of each AU section is given in appendix C.

## 5. INSTRUCTIONS

### a. Scalar Instructions

The scalar instruction group includes an extensive set of load and store instructions: Halfword, fullword, and doubleword instructions, with immediate magnitude, and negative operand capabilities. Ability to load and store register files and to load effective addresses is also available. Arithmetic scalars include various adds, subtracts, multiplies, and divides for halfword (16-bit) and fullword (32-bit) fixed point numbers and fullword and doubleword (64-bit) floating point numbers. Scalar logical instructions include variations of AND, OR, and EXCLUSIVE OR. Shift capabilities for arithmetic and logical operands are provided together with circular shifts. Various comparison instructions and combination comparison-logical instructions are provided for halfwords, fullwords, and doublewords. Many combinations of test and branching instructions with incrementing or decrementing capabilities are also available. Stacking and modifying arithmetic registers can be done with single instructions. Subroutine linkage is accomplished through branch and load instructions. Format conversion for single and double words, as well as normalize instructions, is available.

The scalar instruction word of the Central Processor contains 32 bits and is divided into five fields.

## b. Vector Instructions

The vector repertoire of instructions includes such arithmetic operations as add, subtract, multiply, divide, vector dot product, matrix multiplication and others for both fixed point and floating point representation. Vector instructions are also available for shifting, logical operations, comparisons, format conversions, normalization, and special operations such as Merge, Order, Search, Peak Pick, Select and Replace, among others.

Vector operations can operate on both dimensioned and discontiguous data structures. Up to three dimensions can be defined allowing operations, such as matrix multiplication, to be performed with a single instruction.

A vector instruction word of the CP contains 32 bits and is divided into five fields.

## c. Instruction Timing

The maximum number of clock periods required to execute a scalar instruction is 30; however, the majority of these instructions produces a result in five or less clock periods. The maximum number of clock periods required to produce a vector result is 19; however, the majority of these instructions produces a result in two or less not counting vector setup time. Table 3 gives the typical number of clock periods required to produce a result from the indicated scalar or vector instruction using directly addressed register to register operands to produce a 64 bit result.

Table 3

ASC TYPICAL INSTRUCTION TIMES (CYCLES)<sup>a</sup>

Operation	Scalar	Vector <sup>b</sup>
		Setup Time/Execution Rate (cycle/result)
Floating Add	5	$31+7(N/4)/1.75$
Floating Subtract	5	$31+7(N/4)/1.75$
Floating Multiply	6	$32+3N/3$
Floating Divide	26	$52+19N/19$

<sup>a</sup>1 cycle = 80 ns.

<sup>b</sup>Vector setup time, with no memory conflicts, is 26 cycles + AU scalar time. N is the vector length.

Factors affecting scalar and vector instruction timing are given in appendix D.

## SECTION III

## CRAY-1\*

## 1. OVERVIEW

The Cray Research Corporation CRAY-1 Computer incorporates into its design (1) a Central Processor with separate vector and scalar instructions, (2) up to 1 million 64-bit words of bipolar Central Memory, and (3) an Arithmetic Unit composed of 12 separate functional units.

The configuration shown in figure 14 depicts how the memory and input/output interface to the various CRAY-1 registers, instruction buffers, and functional units.

The primary operating registers of the CRAY-1 are the scalar and vector registers, called S and V registers, respectively. Each of the eight V registers has 64 elements. A scalar instruction may perform some function, such as addition, obtaining operands from two S registers and entering the result into another S register. The analogous vector instruction performs the same function, obtaining at each clock period (12.5 ns) a new pair of operands from two V registers. Results are entered into elements of another V register. The contents of the vector length (VL) register determines the number of operations performed by the vector instruction. Eight 24-bit address (A) registers are used for memory references and as index registers. The A and S registers are each supported by 64 rapid-access temporary storage registers, called B and T registers, respectively. Data can be transferred between the A, B, S, T, or V registers and memory.

All instructions, which may be 16 or 32 bits, are executed from four instruction buffers, each consisting of sixty-four 16-bit registers. Associated with each instruction buffer is a base address register that is used to determine if the current instruction resides in a buffer. Since the four instruction buffers are large, substantial program segments may reside in the buffers. Forward and backward branching within the buffers is possible and the program segments may be contiguous. When the current instruction does not reside in a buffer, one of the

---

\*Reprinted from The CRAY-1 Computer Preliminary Reference Manual and An Introduction to the CRAY-1 Computer, copyrighted 1975, with permission of Cray Research.

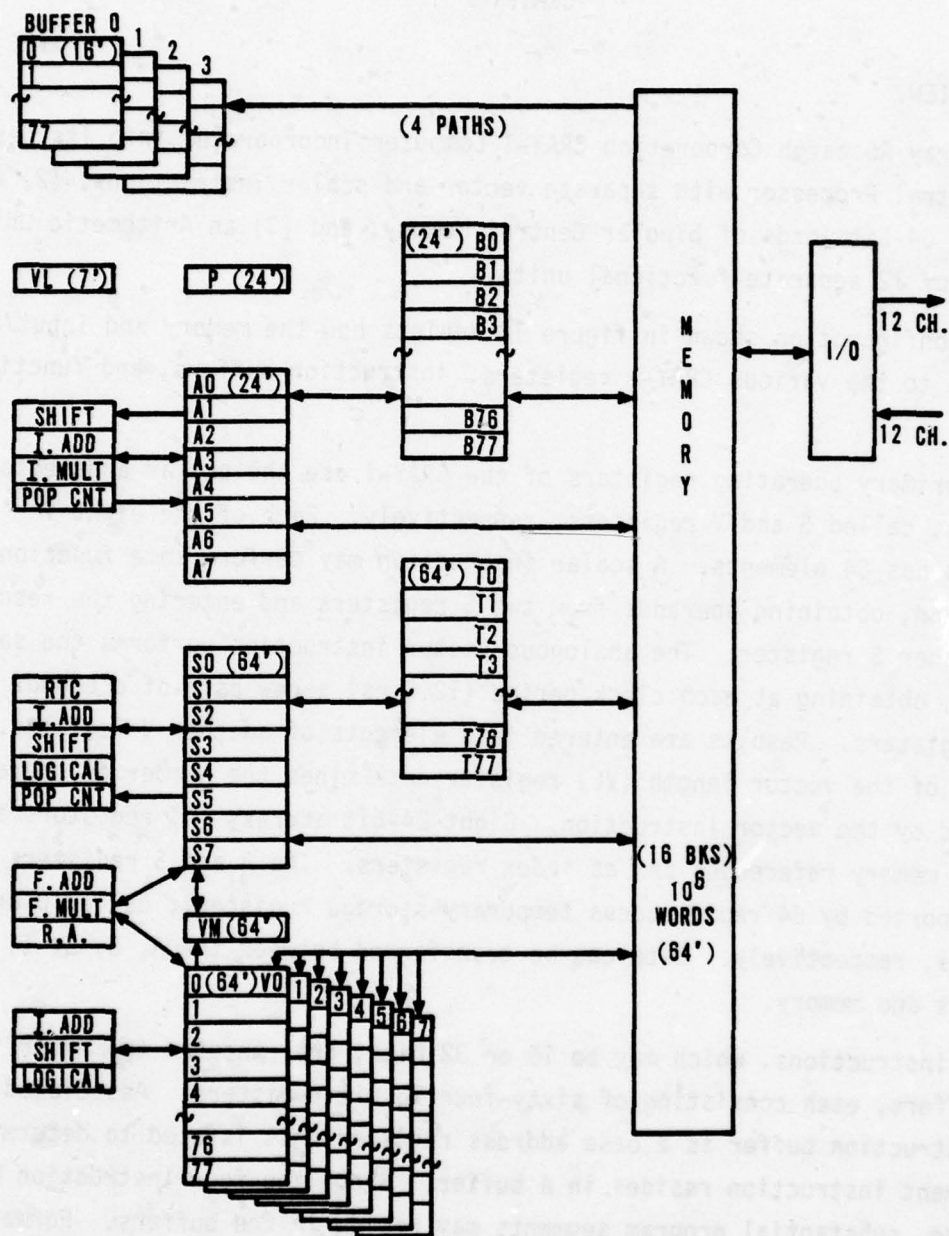


Figure 14. CRAY-1 Central Processor and Memory Diagram

instruction buffers is filled from memory. Four memory words are read per clock period to the least recently filled instruction buffer. To allow the current instruction to issue as soon as possible, the memory word containing the current instruction is among the first to be read.

The CRAY-1 Central Memory is constructed of bipolar 1024 bit-Large Scale Integration (LSI) chips. Up to 1 million 64-bit words are arranged in 16 banks with a bank cycle time of four clock periods or 50 ns. The short cycle time provides an extremely efficient random-access memory. One parity bit per word is maintained in 16 modules of the CPU. There is no inherent memory degradation for machines with less than 1 million words of memory.

There are 24 I/O channels, 12 of which are input and 12 output. Any number of channels may be active at a given time. Each channel has a maximum transfer rate of 640 M bits/sec. At most, one 64-bit word per clock period can be transferred to or from memory, and this is attained when four input channels and four output channels are simultaneously operating at their maximum rate. In practice, this theoretical transfer rate is limited by the speed of peripheral devices and by memory reference activity of the CP.

## 2. CENTRAL PROCESSOR

The Central Processing Unit (CPU) is designed primarily for the execution (issuing) of program instructions in a single instruction pipeline stream. The stream consists of a mixture of 16- and 32-bit scalar (single operands) and vector (array operands) instructions. The CPU has a primary clock period of 12.5 ns and contains 155 principal operating registers. Register functions and designations are listed in table 4 and described below in paragraph 2a.

Table 4

### CRAY-1 REGISTER FUNCTIONS AND DESIGNATIONS

<u>Register Function</u>	<u>Register Designation</u>
8 Base Address Registers	A
64 Temporary Storage Registers	B
8 Scalar Arithmetic	S
64 S Temporary Storage Registers	T
8 64 Element Vector Arithmetic	V
1 Vector Length	VL
1 Vector Mask	VM
1 Parcel Address	P

## a. Registers

## (1) Principal Operating Registers

A registers--The eight 24-bit A registers are primarily used as address registers for memory references and as index registers. They are individually designated by the symbols A0 to A7. Data flow between these registers and the B, S, and VL registers. Data may be directly transferred between the A registers and memory.

B registers--The sixty-four 24-bit B registers provide rapid-access temporary storage for the A registers. They are individually designated by the symbols B0 to B77. Data may be directly transferred between the B registers and memory.

S registers--The eight 64-bit S registers are the principal scalar registers for the CPU. They are individually designated by the symbols S0 to S7. These registers serve as source and designation registers in scalar arithmetic and logical instructions. Data flow between these registers and the A, T, V, and VM registers. Data may be directly transferred between the S registers and memory.

T registers--The sixty-four 64-bit T registers provide rapid-access temporary storage for the S registers. They are individually designated by the symbols T0 to T77. Data may be directly transferred between the T registers and memory.

V registers--The eight 64-element V registers are the operating registers for vector computations. Each element is 64 bits. The V registers are individually designated by the symbols V0 to V7. These registers serve as source and destination registers in vector arithmetic and logical instructions. Data flow between these registers and the S registers. Data may be directly transferred between the V registers and memory.

VL registers--The 7-bit VL register specifies the vector length. Vector computations are performed on vectors of the length specified by the contents of VL.

VM register--The 64 bit VM register contains a vector mask to control register selection in the vector merge instructions. Each bit of the VM register corresponds to a vector element.

P register--The 24-bit P register specifies the parcel address of the current program instruction. The high order 22 bits specify a memory address and the low order 2 bits specify the parcel number.

## (2) Supporting registers

The CPU contains a number of registers which support the operating registers in the execution of programs. These registers are loaded with new information during the execution of an exchange sequence. The information is not altered during the execution interval for an exchange package. These registers are listed below with the description of the individual function performed.

BA register--This 18-bit register holds the base address during the execution interval for each exchange package. The contents of this register are interpreted as the upper 18 bits of a 22-bit memory address. The lower 4 bits of the address are assumed zero. Absolute memory addresses are formed by adding (BA)\* 16 to the relative address specified by the CPU instructions.

LA register--This 18-bit register holds the limit address during the execution interval for each exchange package. The contents of this register are interpreted as the upper 18 bits of a 22-bit memory address. The lower 4 bits of the address are assumed zero. The BA and LA registers together provide memory protection. No memory references may be made below BA nor at or above LA. Such a reference will cause the program or operand range flag to be set and the execution interval of the exchange package will be terminated.

XA register--This 8-bit register holds the upper 8 bits of a 12-bit exchange address during the execution interval for each exchange package. The low order 4 bits of the exchange address are assumed zero.

When the execution interval terminates, the exchange operation exchanges the contents of the registers with the contents of the exchange package at (XA)\*16 in memory. The exchange operation saves the contents of the A, S, P, and VL registers and the supporting registers BA, LA, XA, M, and F.

F register--This 9-bit register contains flags which are set to indicate the conditions causing an exchange operation.

M register--This 3-bit register specifies the modes for generation of interrupts. All interrupts are inhibited when the monitor mode bit is set. Interrupts on storage parity errors are enabled when the storage parity mode bit is set. Interrupts on scalar floating point overflow are enabled when the floating point mode bit is set.

### b. Instruction Buffers

There are four instruction buffers, each consisting of sixty-four 16-bit registers. All instructions are executed from the instruction buffers. An instruction buffer supplies instructions to the next instruction parcel (NIP) and the current instruction parcel (CIP) registers. Associated with each instruction buffer is a base address register that specifies the high order 18 bits of the parcel addresses contained in the instruction buffer. The base address registers are scanned each clock period. If the high order 18 bits of the P register matches one of the base addresses, the proper instruction is selected from the instruction buffer and sent to the NIP register. The instruction is moved to the CIP register for execution. The second parcel of a two-parcel instruction resides in the NIP register when the instruction issues.

When the high order 18 bits of the P register do not match any instruction buffer base address, an "out of buffer" condition exists and instructions are read to an instruction buffer from memory. When an out of buffer condition occurs, the instruction buffer that receives the instruction is determined by the 2-bit counter. Each occurrence of an out of buffer condition causes the counter to be incremented. The first four instruction parcels in an instruction buffer are always from bank 0; however, the first parcels read into an instruction buffer always include the parcel specified by the contents of the P register.

### c. Functional Units

There are 12 functional units in the CPU. Each is a specialized unit implementing algorithms for a portion of the instructions. Each unit is independent of the other units and a number of functional units may be in operation at the same time. A functional unit receives operands from registers and delivers the result to a register when the function has been performed. No information is retained in a functional unit for reference in subsequent instructions. These units operate essentially in three-address mode with limited source and destination addressing.

Three functional units provide the following 24-bit results to the A registers only:

- Integer add
- Integer multiply
- Population count

Three functional units provide the following 64-bit results to the S registers only:

Integer add  
Shift  
Logical

Three functional units provide the following 64-bit results to the V registers only:

Integer add  
Shift  
Logical

Three functional units provide the following 64-bit results to either the S or V registers:

Floating add  
Floating multiply  
Reciprocal approximation

Integer arithmetic is performed in two's complement mode. Floating point quantities have signed magnitude representation.

All functional units are fully segmented or pipelined. This means that the information arriving at each unit, or moving within a unit, is captured and held in a new set of registers at the end of every clock period. It is therefore possible to start a new set of operands for unrelated computation into a functional unit each clock period even though the unit may require more than one clock period to complete the calculation. All functional units perform their algorithms in a fixed amount of time. No delays are possible once the operands have been delivered to the unit. Functional units servicing the vector instructions produce one result per clock period. In general, the number of operations which can be performed simultaneously by a functional unit is equal to the functional unit time in clock periods.

#### d. Functional Unit and Operand Register Reservations

When a vector instruction issues, the required functional unit and the operand registers are reserved for the number of clock periods determined by the vector length. A subsequent vector instruction which requires the same functional unit or operand register cannot issue until the reservations are released. When two vector instructions use different functional units and vector registers, they are independent and may issue in neighboring clock periods. Example (1) shows two independent instructions. Both execute concurrently with

a one clock period difference in their issue times. Examples (2) through (4) illustrate the effect of functional unit and operand register reservation when two instructions are not independent. Example (2) shows two add instructions. When the first instruction issues, the floating add functional unit and operand registers V1 and V2 are reserved. Issue of a second add instruction is delayed until the functional unit is free. Example (3) shows an add instruction followed by a multiply instruction. As in the previous example, the floating add functional unit and operand registers V1 and V2 are reserved when the first instruction issues. Issue of the second instruction is delayed until the operand register V1 is free. The second instruction in example (4) is delayed because of both functional unit and operand register reservations.

Examples:

- |   |  |
|---|--|
| $\begin{array}{l} V0 \leftarrow V1 + V2 \\ V3 \leftarrow V4 * V5 \end{array}$ <p>(1) Independent Instructions</p> $\begin{array}{l} V3 \leftarrow V1 + V2 \\ V6 \leftarrow V1 * V5 \end{array}$ <p>(3) Operand register reservation</p> | $\begin{array}{l} V3 \leftarrow V1 + V2 \\ V6 \leftarrow V4 + V5 \end{array}$ <p>(2) Functional unit reservation</p> $\begin{array}{l} V0 \leftarrow V1 + V2 \\ V3 \leftarrow V1 + V5 \end{array}$ <p>(4) Functional unit and operand register reservation</p> |
|---|--|

### 3. INPUT/OUTPUT

There are 24 I/O channels. The channels are assigned the numbers 2 through 25. Each input channel consists of a data channel (16 data bits and 3 control bits), a 64-bit assembly register, a current address (CA) register, and a channel limit address (CL) register. Each input channel can cause a CPU interrupt condition when the current address equals the limit address register value or when the input device sends a disconnect.

Each output channel consists of a data channel (16 data bits and 3 control bits), a 64-bit disassembly register, a CA register, and a CL register. Each output channel can cause a CPU interrupt condition when the current address equals the limit address register value. A disconnect is sent on the output channel after the last word of a record is sent and acknowledged.

#### 4. INSTRUCTIONS

##### a. Scalar Instructions

Scalar instructions apply a function to one or two operands in registers and enter the result into a register. The addition of two integers in S1 and S2, entering the sum into S3, is an example of a scalar instruction.

Four conditions must be satisfied for issue of a scalar instruction:

The functional unit must be free. No conflicts can arise with other scalar instructions; however, vector floating point instructions reserve the floating point units. Memory references may be delayed due to conflicts.

The result register must be free.

The operand registers must be free.

The result register group input path must be free at execution time - 1 cycles. One input path exists for each of the four register groups (A,B,S, and T).

Scalar instructions place reservations only on result registers. A result register is reserved for the execution time of the instruction. No reservations are placed on the functional unit or operand registers.

The scalar instruction set includes load and store instructions and interregister transmissions. Also available are A register integer sum, difference, and product, and S register integer/floating point sum and difference, and floating point product. Several branch-on-conditions of register S as well as various shifts and complements are provided.

##### b. Vector Instructions

The vector instruction set is comprised of logical sum, difference, and product between various combinations of S and V registers. In addition, shifts and double shifts; integer sum and difference; and floating point sum, difference, and product also exist in the vector instruction list.

Vector instructions may be classified into four types. One type of vector instruction obtains operands from one or two V registers and enters results into another V register (figure 15). Successive operand pairs are transmitted from Vj and Vk to the segmented functional unit each clock period, and the corresponding results emerge from the functional unit n clock periods later (n is constant for a given functional unit and is called the functional unit time). Results are entered into result register Vi. The contents of the VL register determine the number of operand pairs processed by the functional unit.

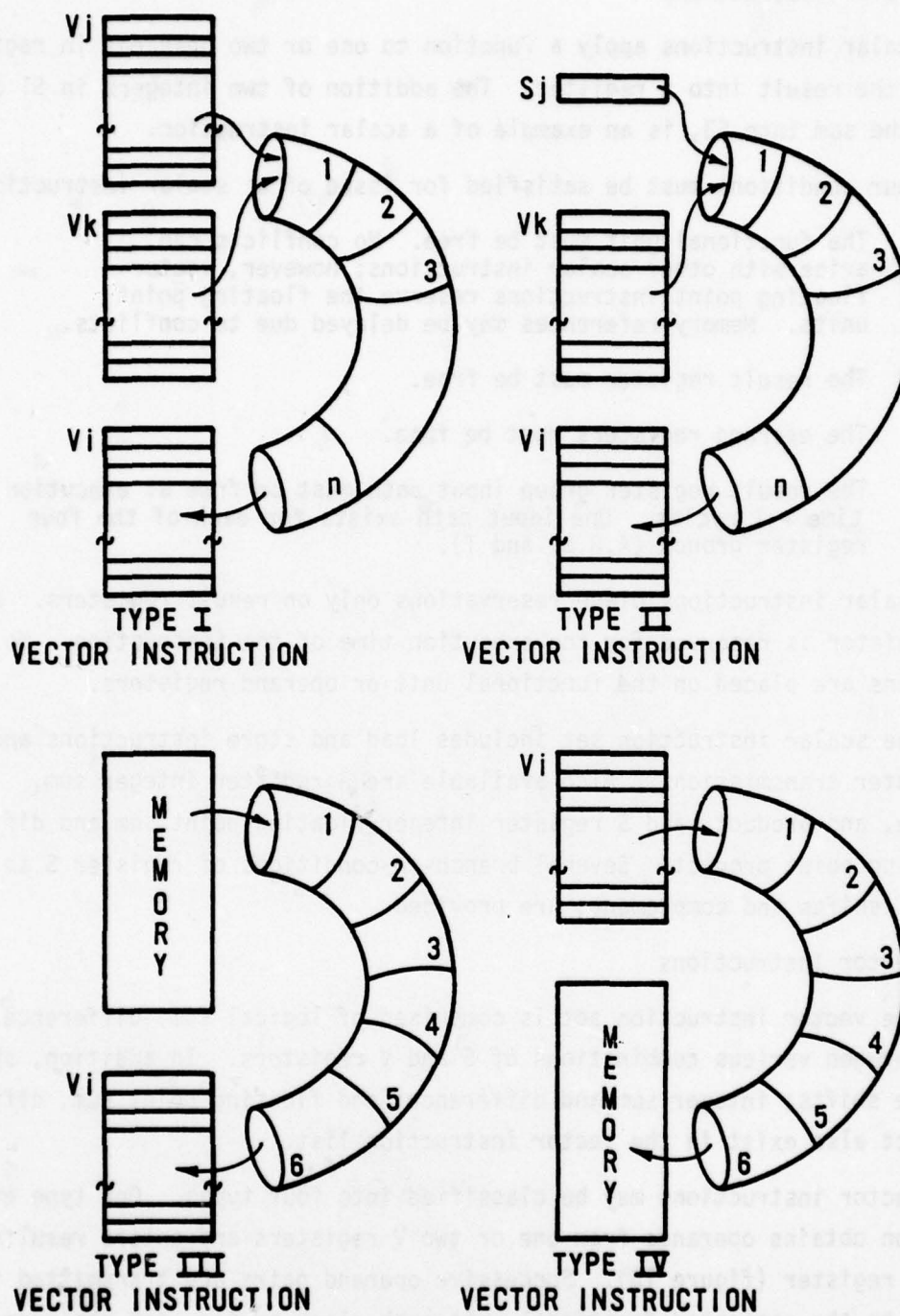


Figure 15. CRAY-1 VECTOR INSTRUCTION TYPES

The second type of vector instruction obtains one operand from an S register and one from a V register (figure 15). A copy of the S register is transmitted to the functional unit with each V-register operand.

The last two types of vector instruction transmit data between memory and the V registers (figure 15). A path between memory and the V registers may be considered a functional unit for timing considerations. Four conditions must be satisfied for issue of a vector instruction:

The functional unit must be free.

The result register must be free.

The operand registers must be free or at chain slot time.

Memory must be quiet if the instruction references memory.

Vector instructions place reservations on functional units and registers for the duration of execution as described below:

Functional units are reserved for VL+2 clock periods except for two special cases:  
 -Memory is reserved for VL+4 clock periods.  
 -A shared functional unit is reserved for VL+4 clock periods if a subsequent scalar instruction requires the unit

The result register is reserved for the functional unit time + (VL+2) clock periods. The result register is reserved for the functional unit time +7 clock periods if the vector length is less than 5. At functional unit time +2 (called chain slot time) a subsequent instruction, which uses the reserved result register as an operand register and which has met all other issue conditions, may issue. This process is called "chaining." Several instructions using different functional units may be chained in this manner to attain a significant enhancement of processing speed.

Vector operand registers are reserved for VL+1 clock periods. Vector operand registers are reserved for 6 clock periods if the vector length is less than 5. The vector register used in a block store to memory is reserved for VL+5 clock periods. Scalar operand registers are not reserved.

It is important to understand functional unit segmentation, or pipelining in the CRAY-1, especially as it relates to execution of vector instructions. Let a particular element of a V register be specified by adding the element number

as a subscript to the register name. For example, the first three elements of register V1 are  $V1_0$ ,  $V1_1$ , and  $V1_2$ , respectively. Since a vector register has 64 elements, the last element of V1 is  $V1_{63}$ . Figure 16 shows a timing chart for execution of a floating point addition instruction. This instruction is type 1 since operands are obtained from two vector registers. When the instruction issues at clock period  $t_0$ , the first pair of elements ( $V1_0$  and  $V2_0$ ) is transmitted to the add functional unit where it arrives at clock period  $t_1$ . The dashed lines indicate transit to and from the functional unit. The functional unit time for this unit is 6 clock periods, so the first result, which is the sum of  $V1_0$  and  $V2_0$ , exits from the functional unit at clock period  $t_7$ . The sum is transmitted to the first element of result register V0 arriving at clock period  $t_8$ . Because the functional unit is fully segmented, the second pair of elements ( $V1_1$  and  $V2_1$ ) is transmitted to the add functional unit at clock period  $t_1$ . At clock period  $t_2$  the functional unit is in the process of performing two additions simultaneously, since the addition of  $V1_0$  and  $V2_0$  was begun in the previous clock period. The second result which is the sum of  $V1_1$  and  $V2_1$ , is entered into the second element of result register V0 at clock period  $t_9$ . Continuing in this manner, a new pair of elements enters the functional unit each clock period and the corresponding sum emerges from the unit 6 clock periods later and is transmitted to the result register. Since a new addition is begun each clock period, six additions may be in progress at one time.

The vector length determines the total number of operations performed by a functional unit. Although each vector register has 64 elements, only the number of elements specified by the vector length register is processed by a vector instruction. Vectors which have more than 64 elements are processed under program control, in groups of 64 (with a possible residue). The program construct created to process long vectors is called a vector loop. Each pass through the loop processes a 64-element (or smaller) segment of the long vectors. The general procedure is to compute the loop count based on the vector length before entering the loop. Inside the loop the program takes full advantage of the 12 independent functional units and chaining (described below) to read the current vector segments from memory, execute the required functions, and return the results to memory. Loop control is performed in the scalar registers concurrently with vector processing. Loop branch time is hidden by the vector operations because it is done in parallel with the segment store operation.

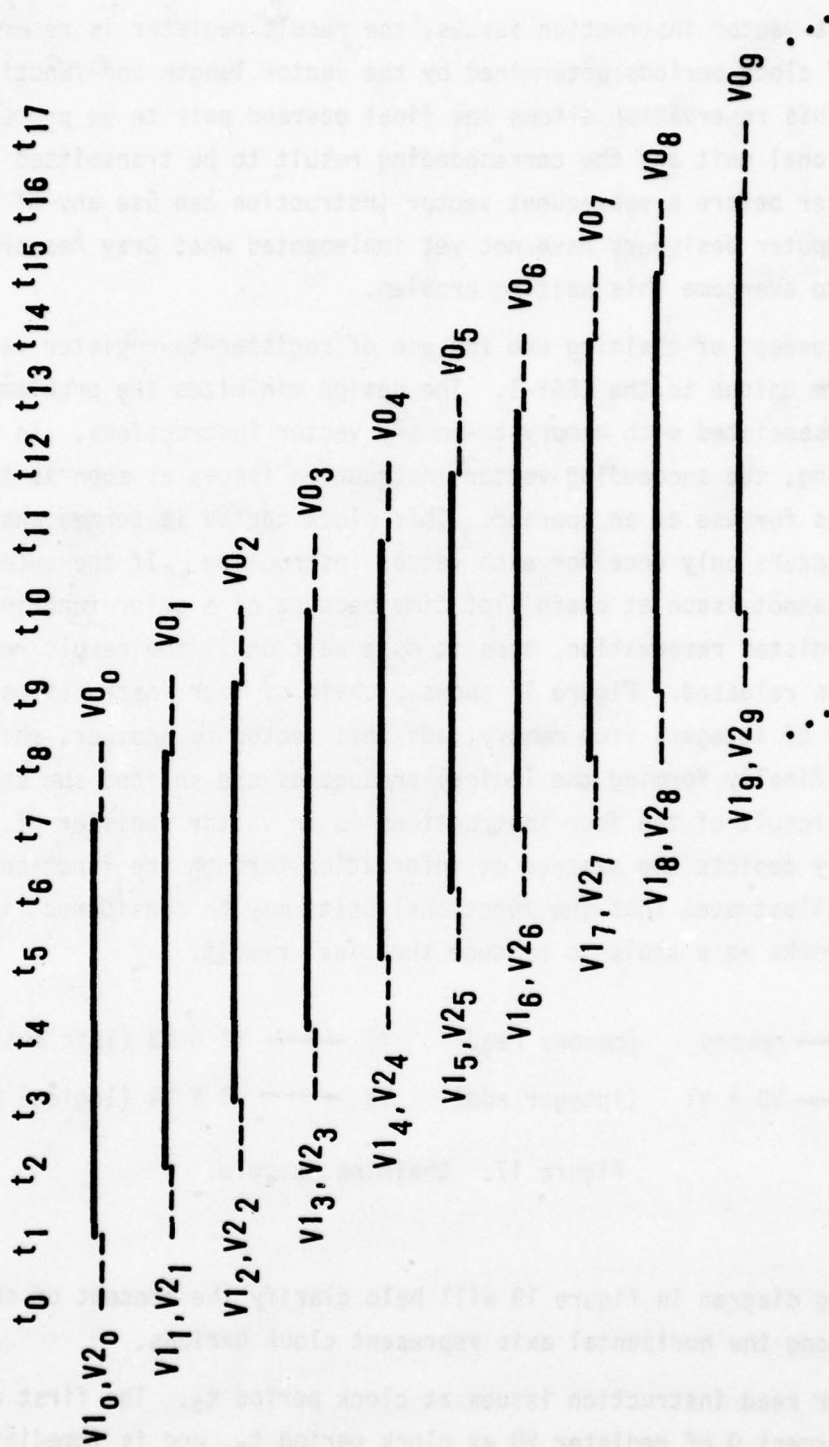


Figure 16. CRAY-1 Vector Instruction Timing Example ( $V0 \leftarrow V1 + V2$ )

## c. Chaining

When a vector instruction issues, the result register is reserved for the number of clock periods determined by the vector length and functional unit time. This reservation allows the final operand pair to be processed by the functional unit and the corresponding result to be transmitted to the result register before a subsequent vector instruction can use any of the results. Computer designers have not yet implemented what Cray Research labels as chaining to overcome this waiting problem.

The concept of chaining and the use of register-to-register vector instructions are unique to the CRAY-1. The design minimizes the problem of speed degradation associated with memory-to-memory vector instructions. In the process called chaining, the succeeding vector instruction issues as soon as the first result arrives for use as an operand. This clock period is termed chain slot time and it occurs only once for each vector instruction. If the succeeding instruction cannot issue at chain slot time because of a prior functional unit or operand register reservation, then it must wait until the result register reservation is released. Figure 17 shows a chain of four instructions which read a vector of integers from memory, add that vector to another, shifting the sum, and finally forming the logical product of the shifted sum and a mask vector. The result of the four instructions is in vector register V5. Figure 18 graphically depicts the passage of information through the functional units. The diagram illustrates that the functional units may be considered links in a chain which works as a whole to produce the final result.

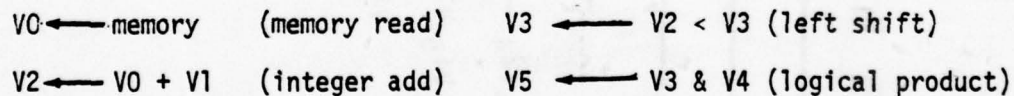


Figure 17. Chaining Example.

The timing diagram in figure 19 will help clarify the concept of chaining. Gradations along the horizontal axis represent clock periods.

The vector read instruction issues at clock period  $t_0$ . The first word arrives in element 0 of register V0 at clock period  $t_8$ , and is immediately transmitted along with element 0 of register V1, as an operand to the integer add functional unit. When the two operands arrive at the integer add functional

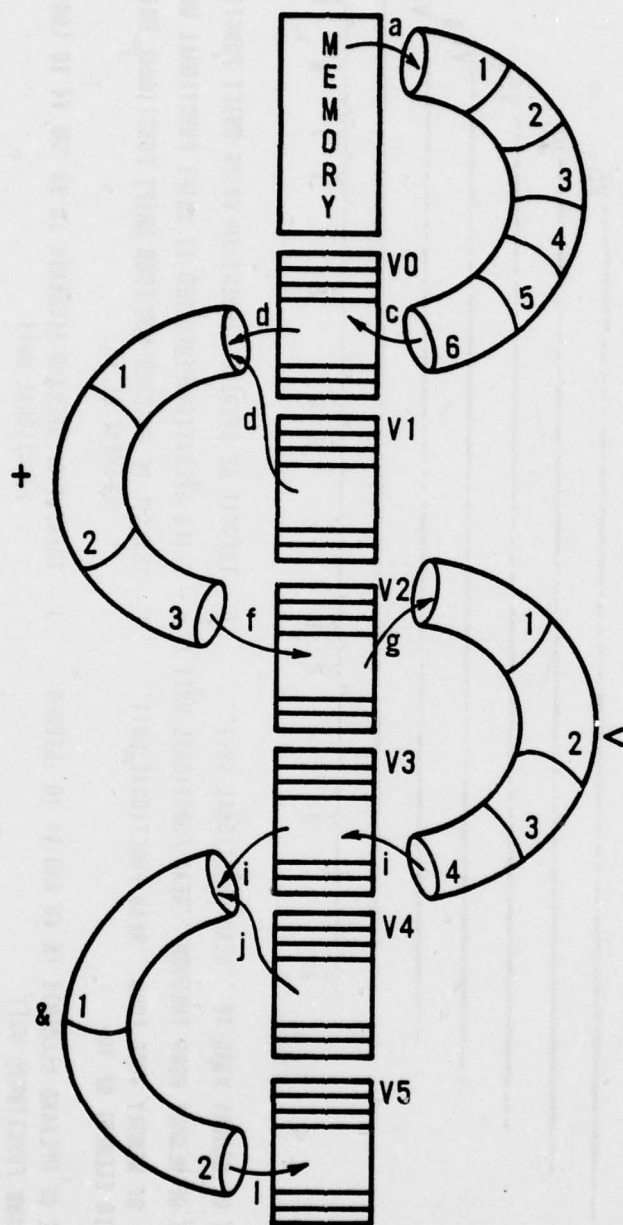


Figure 18. CRAY-1 Pictorial Representation of Chaining Example

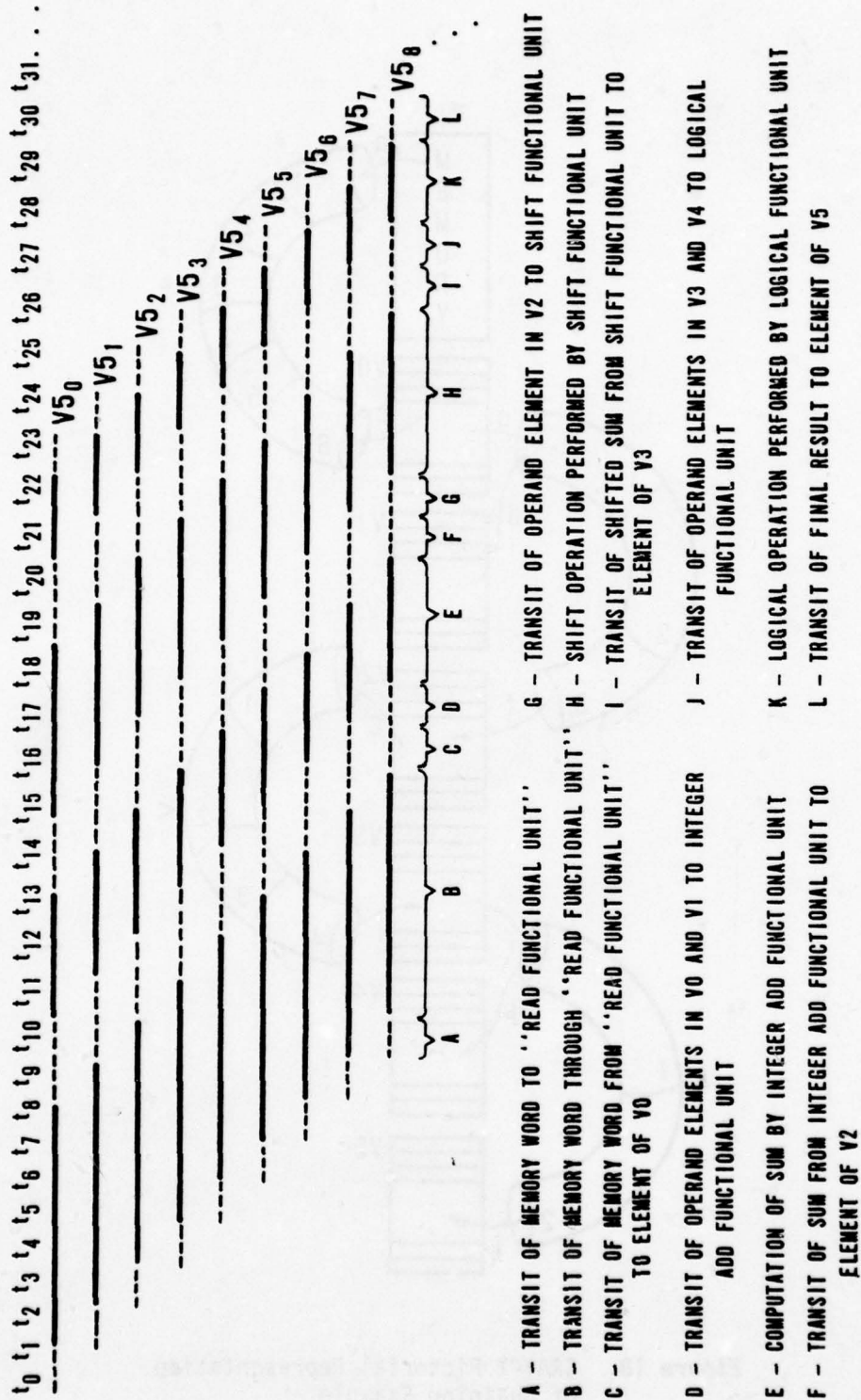


Figure 19. CRAY-1 Timing Diagram for Chaining Example

unit at clock period  $t_9$ , the computation of the sum of  $V0_0$  and  $V1_1$  is begun. Three clock periods later ( $t_{12}$ ) the sum is sent from the functional unit to element zero of  $V2$ . It arrives at clock period  $t_{13}$  and is immediately transmitted as an operand to the shift functional unit. At clock period  $t_{14}$  the operand arrives at the shift functional unit and the shift operation is begun. The operation is completed four clock periods later ( $t_{18}$ ) and the shifted sum is sent from the functional unit to element zero of  $V3$ , arriving the next clock period. It is immediately transmitted, along with element zero of  $V4$ , as an operand to the logical functional unit. When the two operands arrive at the logical functional unit at clock period  $t_{20}$ , the computation of the logical product of  $V3_0$  and  $V4_0$  is begun. Two clock periods later ( $t_{22}$ ) the final result is sent from the functional unit to element zero of  $V5$ , arriving at clock period  $t_{23}$ . While all this has been going on, production of the second element of  $V5$  has been tracing the same path through the vector registers and functional units with a one clock period lag. Production of the third element of  $V5$  lags one more clock period behind, and so on. A new result arrives at the  $V5$  result register each clock period.

#### d. Instruction Formats

There are five instruction formats for the CRAY-1. Each instruction is either a one-parcel (16-bit) instruction or a two-parcel (32-bit) instruction. Two-parcel instructions may begin in the fourth and last parcel position within a word and end in the first parcel position of the next word. The assembler lists a parcel address as a word address followed by a one-character alphabetic (a-d) parcel identifier.

#### e. Instruction Timing

When instruction issue conditions are satisfied, an instruction completes in a fixed amount of time. Instruction issue may cause reservations to be placed on a functional unit or registers. Knowledge of the issue conditions, instruction execution times and reservations permit accurate timing of code sequences. Memory bank conflicts due to I/O activity are the only element of unpredictability.

The maximum number of clock periods required to execute a scalar instruction yielding a 24-bit result is 10 cycles; however, 90 percent of these instruction types produce a result in six or less clock periods. To produce a 64-bit scalar result, a maximum of 14 cycles is required; however, 90 percent of these instruction types produce a result in seven or less clock periods.

The maximum number of clock periods required to produce a 64-bit vector result is 14; however, the majority of these instructions produces a result in seven or less clock periods. Table 5 gives the typical number of clock periods required to produce a 64-bit result from the indicated scalar and vector instructions.

Table 5  
CRAY-1 TYPICAL INSTRUCTION TIMES (CYCLES)<sup>a</sup>

<u>Operation</u>	<u>Scalar</u>	Vector <sup>b</sup>
		<u>Set-up Time/Execution Rate</u> <u>(Cycle/Result)</u>
Floating Add	6	16 + N/1
Floating Multiply	7	17 + N/1
Floating Divide <sup>c</sup>	29	39 + N/1
Reciprocal Approximation	14	24 + N/1

<sup>a</sup>1 cycle = 12.5 ns

<sup>b</sup>Vector length is N

<sup>c</sup>No single instruction can perform the divide function. Divide is made up of the Reciprocal Approximation and three Multiply Instructions.

## SECTION IV

## STAR-100\*

## 1. OVERVIEW

The Control Data Corporation STAR-100 (SString ARray) computer is a high-speed, logical, and arithmetic computer providing multiprogrammed operation on batch and interactive job streams. The STAR-100 contains stream arithmetic and functional units especially designed for sequential and parallel operations on single bits, 8-bit bytes, and 32-bit or 64-bit floating point operands and vectors. The STAR-100 has a clock period of 40 ns and uses the concepts of virtual memory operation and distributive and string array processing.

The basic STAR-100 Computer consists of the Central Processor Unit (CPU), 524,288 64-bit words of memory, four input/output channels, and a Maintenance Control Unit (MCU). The number of input/output channels can be expanded to 12, in increments of four. An additional 524,288 words of memory may be added for a maximum memory size of 1,048, 576 words. Figure 20 shows the basic computer configuration.

The CPU contains functions of storage access control (SAC), stream, string, and floating point. The SAC unit controls I/O channels, data transmission to and from memory, memory parity checking and virtual addressing comparison and translation. The stream unit performs all streaming and instruction control, operand alignment, buffering, and addressing. This unit contains a 64-bit by 256-location register file which is used for instruction and operand addressing,

---

\*Reprinted from "Control Data STAR-100 Computer Hardware Reference Manual," Pub No. 6025600, copyright 1970, 1971, 1973, 1974, 1975 by Control Data Corporation; "Control Data STAR Peripheral Stations Hardware Reference Manual," Pub No. 60405000, copyright 1973, 1974, 1975 by Control Data Corporation; "Control Data STAR Computer System Operating System Reference Manual," Pub No. 60384400, copyright 1973, 1974, 1975, 1976 Control Data Corporation; "Control Data STAR Computer System STAR Assembler Reference Manual," Pub No. 19980200, copyright 1973, 1974, 1976 Control Data Corporation; "Control Data STAR-100 Computer Features Manual," Pub No. 60425500, copyright 1973, 1974, 1975 by Control Data Corporation. Timing information and pipe operation reprinted from "Control Data STAR-100 Computer Abbreviated Operand Processing and Basic Instruction Timing Information" with permission of Control Data Corporation.

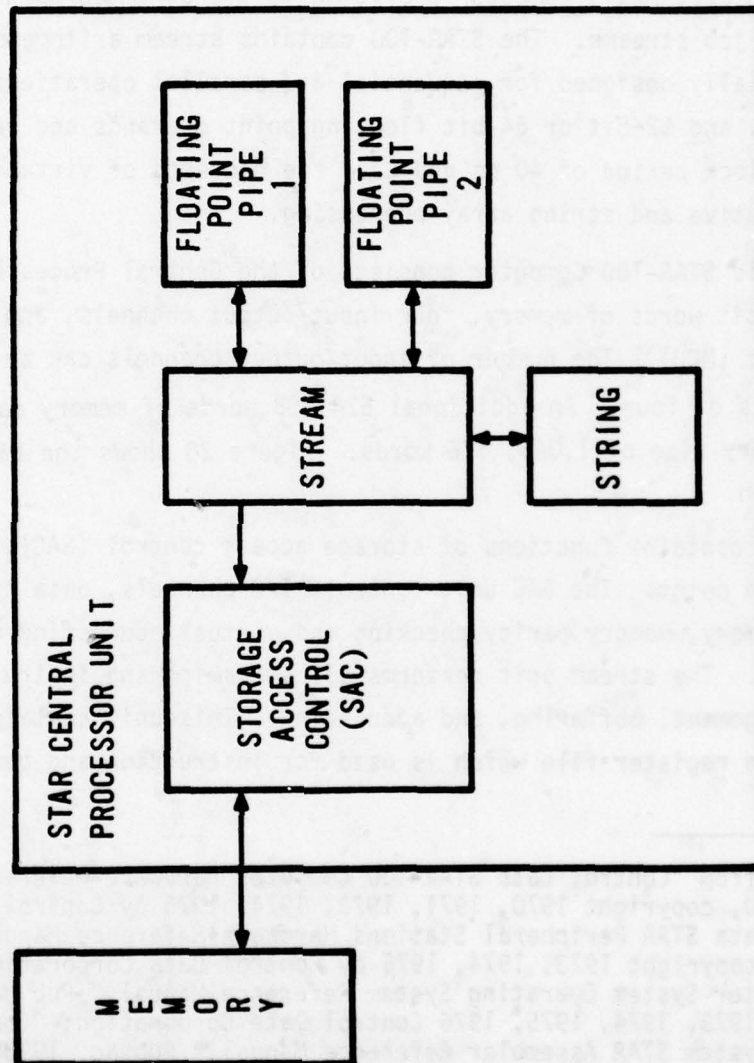


Figure 20. STAR-100 Central Processor

indexing, field length counts, and source and destination points for register instruction operands and results. A microcode memory in the stream unit controls setup, interrupt, and termination of vector-like instructions. The string and floating point units perform the majority of the computer arithmetic operations.

The I/O channels consist of control units for 16-bit data communications between the SAC and the MCU and between the SAC and peripheral stations. Any one of the I/O channels connects to the MCU, and the other channels connect to the peripheral stations. These stations consist of a buffer controller and related control circuitry connected to the corresponding peripheral equipment.

The virtual memory and associated paging scheme of the STAR manages allocation of storage between main memory and auxiliary memory, moves information from auxiliary to main memory as needed, and translates virtual memory addresses to physical addresses in main memory. Every program is considered to be executable only in virtual memory. Data files may be defined either by a set of virtual addresses or by physical mass storage addresses, and will be translated to appropriate virtual addresses.

The internal structure and organization of the operating system fosters efficient processing of string array and vector instructions. In floating point operations, vector concepts and the streaming of vector information through STAR's two parallel pipeline processors allow several programs to be in various stages of execution at the same time. The two pipeline units operate on strings of operands, 64- or 32-bit arrays, byte strings, and bit strings. Information to specify the addresses of source and destination streams usually is held in the stream unit register file. Core storage system design accommodates two input operand streams and one output stream simultaneously.

The STAR-100 string instructions can operate on a maximum of 65,536 operands in one pass. Although the function is executed serially in a pipeline, it may be considered as being carried out in parallel on the data. Processing facilities allow for efficient computing in the conventional sense and also provide a fundamentally different approach to programming through string and array processing.

The computation unit of the STAR-100 is an autonomous central processor with input/output channel connections. Stations, consisting of a minicomputer, display/keyboard unit, small drum and buffer memory, handle peripheral processing functions and are linked to the STAR central processor. Slow speed input/output devices, terminals, magnetic tapes, etc., are grouped and connected to stations to comprise a distributed system.

## 2. CENTRAL MEMORY

### a. Main Memory

The Central Memory (CM) of the STAR-100 is composed of magnetic core storage. Magnetic Core Storage (MCS) consists of 524,288 66-bit words (64 data bits and 2 parity bits), physically arranged as 65,536 528-bit words. For convenience, the MCS is referred to as a 525K memory (for 66-bit words) and a 65K memory (for 528-bit words). Each 528-bit word is called a super word or sword and is contained in two 264-bit planes. The MCS is divided into 32 banks, physically located in eight sections. For addressing considerations, one bank contains 2048 addresses of 528 bits each. Two planes are referenced simultaneously to read or write one 528-bit sword. An MCS option for another 524K memory may be added to the computer. The MCS option requires an additional eight sections of MCS.

A storage word in CM is considered as one sword (figure 21). One sword contains four quarter-swords. Each quarter-sword in turn contains two 66-bit words which are addressed from left to right within the sword. Each 64-bit word can be further broken down into an upper and lower portion each consisting of 32-bits.

The 528 bits of one sword transfer to/from MCS during each write/read operation, although only part of the sword may actually be stored or used. When the SAC performs a write/read operation, it addresses each of the eight MCS sections. In addition, SAC sends a bank request signal that selects only one of the 32 memory banks. A storage word transfer then takes place between SAC and the selected MCS memory bank. The transfer occurs in four quarter-sword transmissions. The transmissions go through a 132-bit data trunk to the MCS section that contains the selected memory bank. The transfer requires a period of four cycles, one quarter-sword per cycle. During a write operation, SAC sends a write enable signal for each half-word (32-bits). Depending on how the enables are set, any or all of the half-words within the sword may be written into storage. The SAC unit sends the enable signals in two 8-bit groups. Similarly, SAC may select and use any or all of the half-words of a sword that it receives in a read operation. Data parity checking and generating are accomplished in SAC.

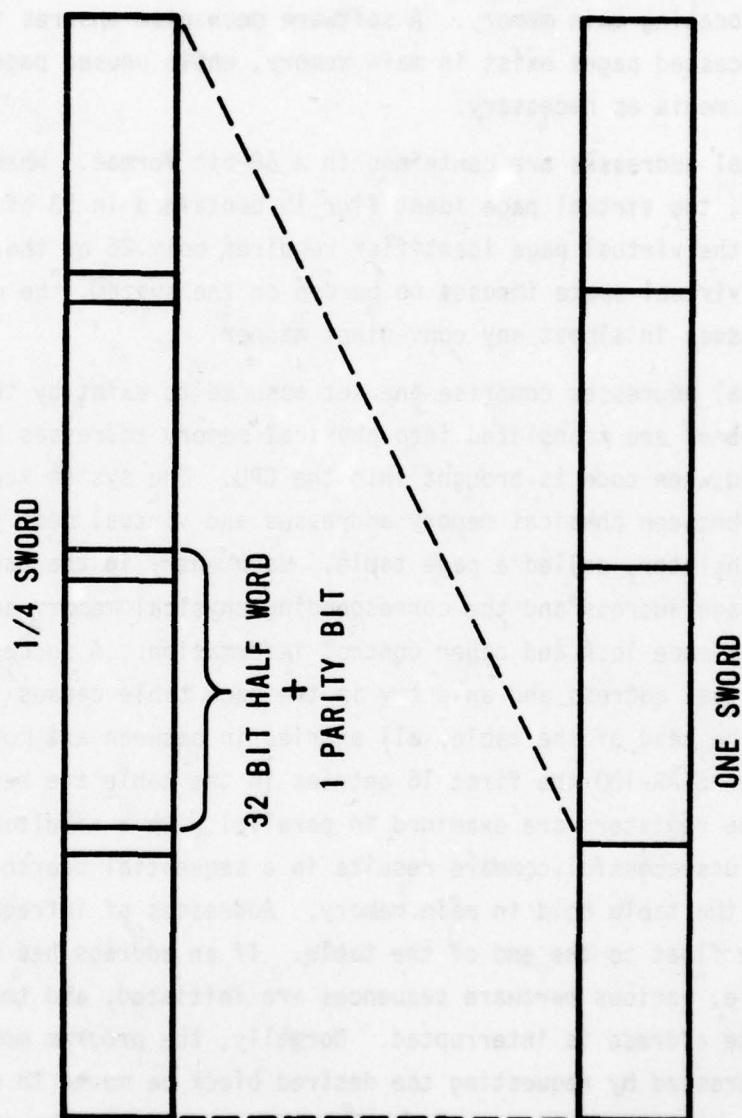


Figure 21. STAR-100 Super Word

### b. Virtual Memory

Through the virtual memory system of the STAR-100, an apparently unlimited memory structure may be viewed as if it were entirely main memory. The STAR-100 hardware mechanisms manage system information in 65,536-word blocks (large pages) or in 512-word blocks (small pages). System software determines the block size to use in allocating main memory. A software mechanism ensures that the most frequently accessed pages exist in main memory, while unused pages are sent to slower backup media as necessary.

Virtual addresses are contained in a 48-bit format. When 512-word pages are addressed, the virtual page identifier is contained in 33 of the 48 bits; for large pages, the virtual page identifier requires only 26 of the 48 bits. Because unused virtual space imposes no burden on the system, the user may organize program addresses in almost any convenient manner.

Virtual addresses comprise the set assumed to exist by the programmer. Virtual addresses are translated into physical memory addresses by system software as needed when code is brought into the CPU. The system keeps track of the relationship between physical memory addresses and virtual memory addresses through a translator, called a page table. Each entry in the page table contains the virtual page address and the corresponding physical memory address, together with an access mode lock and other control information. A successful association between a virtual address and an entry in the page table causes that entry to be moved to the head of the table; all entries in between are moved down by one place. In the STAR-100 the first 16 entries in the table are kept in high speed registers; the registers are examined in parallel with a simultaneous associative compare. An unsuccessful compare results in a sequential search through the remainder of the table held in main memory. Addresses of infrequently used pages automatically float to the end of the table. If an address has no entry in the page table, various hardware sequences are initiated, and the program requesting the address is interrupted. Normally, the program monitor provides the space addressed by requesting the desired block be moved to main memory either from a storage station or from the paging station. The program is restarted, to continue processing from the point of interruption.

### 3. CENTRAL PROCESSOR

The CPU shown in figure 20 consists of the following functional areas: storage access control (SAC); stream unit; string unit; and floating-point unit.

#### 4. Storage Access Control

The SAC unit controls the transmission of data to/from MCS and performs virtual address comparison and translation. The SAC unit also generates parity bits for write data and checks parity for read data. Thus, SAC provides access to MCS for stream and the input/output (I/O) channels.

The SAC unit shown in figure 22 connects to memory via eight read and write data sets. In this case a data set is defined as a physical grouping of cables and associated circuits used to carry data. There is one data set to each memory section. If the optional MCS is connected to the system, a total of 16 data sets is available for data transmission to/from MCS. For each reference, the data transmission to/from MCS are in the form of four 132-bit portions of the 528-bit superword (sword) contained in MCS. Each 132-bit portion is referred to as a quarter-sword and consists of 128 data bits and four parity bits (figure 21). One parity bit is associated with each half-word of data. The SAC unit references memory on a sword basis. In a write operation, write enables determine the number of half-words written into memory. Therefore, less than one sword may be written into memory, even though the time allocation is for a full sword.

A more detailed description of the read/write operations and parity checks performed by the SAC is contained in appendix E.

##### (1) Virtual Address Mechanism

The SAC unit contains 16 associative registers (appendix H) and corresponding control circuits. When the CPU is in job mode (appendix F), all addresses sent from the stream unit are virtual addresses. The SAC unit compares a virtual address with the virtual address identifier of the associative registers. If a match is found and one of four keys compares with the lock of the associated word, the virtual address control circuits convert the virtual address into the corresponding absolute memory address from which the reference is made (appendix H). If no match is found in the associative registers, the virtual address control circuits read additional associative words from a restricted portion of MCS, termed the space table. The associative registers and the space table make up the page table (appendix I). All of the associative registers are compared in 40 ns. Should the compare tests not be met in the registers, the search continues in the space table portion of central memory at the rate of 20 ns per entry.

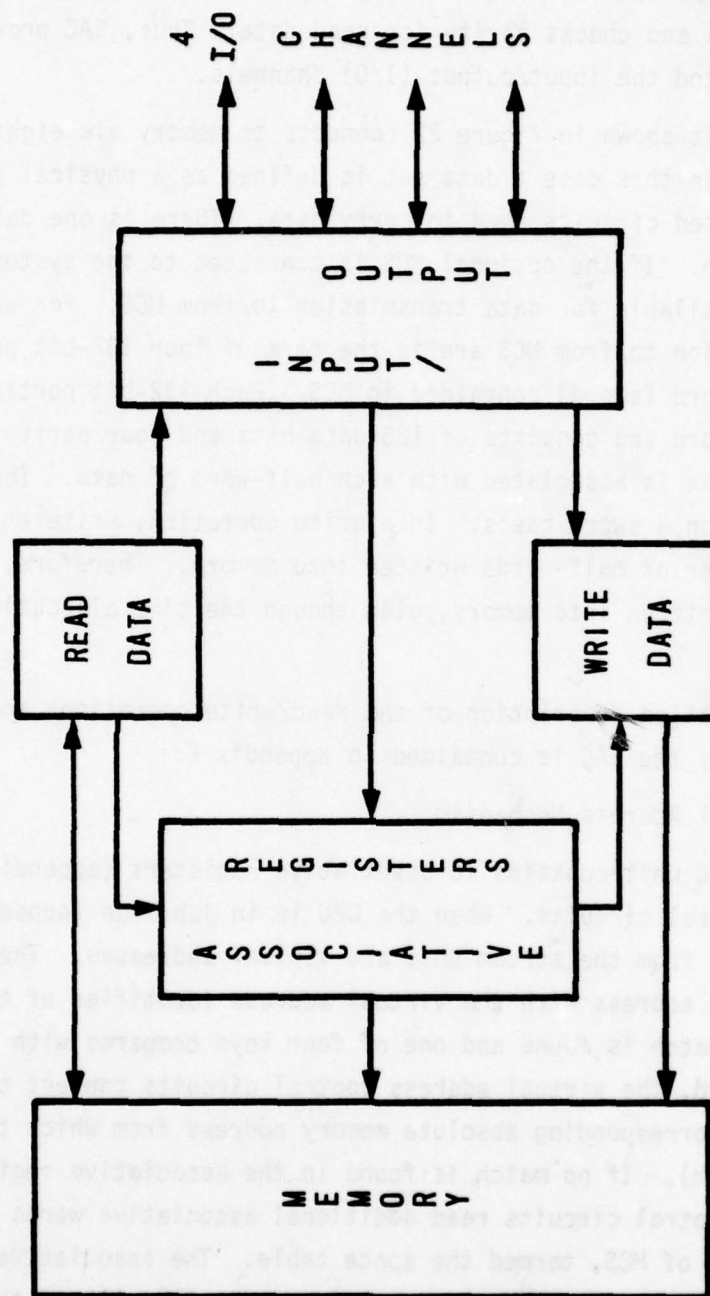


Figure 22. STAR-100 Storage Access Control Unit

## (2) Storage Protect Features

The SAC unit contains the storage protection circuits for the computer system. The storage protection features consist of a lock and key arrangement. Each associative word in the page table contains a 12-bit lock code. The lock code is associated with a page of MCS. Each job is assigned four 12-bit keys by the monitor program. If a virtual address matches the corresponding portion of the associative word, the four keys associated with the current job are compared with the lock code in the matching associative word. One of the four keys must match the lock code before the storage reference can be completed. Thus, the monitor program can restrict MCS page access to only the specified jobs by assigning the lock and key codes accordingly.

In addition to the lock/key protection feature, each of the four keys is associated with a 4-bit usage lockout code. This code can lock out CPU write operations, CPU read operations, and/or CPU instruction references. If a key matches the lock of an associative word, but the requested type of reference is inhibited by the usage lockout code, an access interrupt takes place to the monitor program. Thus, the monitor program can restrict MCS page access for a job to a particular type of reference.

Since during monitor mode (appendix F) all CPU references are absolute addresses, the storage protection features are disabled for these references. In the same manner, I/O channel references are absolute addresses and are unrestricted by the storage protection features.

## (3) Input/Output Channels

There may be up to 12 channels in the CDC STAR-100 SAC unit. Channels 1 through 4 are required in the minimum system and channels 5 through 8 and 9 through 12 may be added as options. One channel must be reserved for the MCU.

A typical I/O channel connects to a peripheral station. The peripheral station may, in turn, be connected to various peripheral devices or be connected to another second-level peripheral station.

Data are transmitted to/from the I/O channel in 16-bit transmissions. In I/O write operations, two successive 16-bit data transmissions from the peripheral station are assembled into one 32-bit half-word. The half-words are temporarily stored in the I/O buffer. When sufficient data have been assembled and stored in the I/O buffer, they are transmitted through the SAC data circuits

to central storage. In I/O read operations, the SAC data circuits transmit one complete sword from central storage into the I/O buffer. The I/O control circuit then reads 32-bit half-words from the I/O buffer into the data registers. The data are disassembled into 16-bit transmissions which are sent to the peripheral station.

At the beginning of an I/O read or write operation, a starting address is sent to the I/O channel in the form of two successive 16-bit transmissions (only 21 of the 32 bits are used). Of the total, 11 bits are used as the MCS sword address and 6 bits are used as the bank address. The remaining bits define the quarter-sword and the half-word addresses for the I/O buffer assembly/disassembly operation. A description of I/O assembly is given in appendix G.

#### b. Stream Unit

The stream unit provides basic control for the computer. It contains the register file, instruction control unit, input/output stream units, stream buffers, and microcode memory. All memory requests from the central processor as well as many control signals originate in this unit. The stream unit performs the following functions:

- Initiates all central storage reference requests for instructions and operands.

- Translates these instructions and transmits control signals to the arithmetic units.

- Provides addressing for all source operands and arithmetic results.

- Buffers and positions all operands and arithmetic results between central storage and the arithmetic units.

- Performs logical instructions such as exclusive OR, AND, inclusive OR, and shift on operands from the register file.

- Performs binary and decimal arithmetic operations on byte strings. It also performs other bit or byte string type operations such as edit, pack, unpack, compare, merge, modulo arithmetic, logical, and search with or without delimiter.

The following paragraphs describe the main functional areas of the stream unit.

#### (1) Instruction Control

Instruction control receives all instructions from central storage via a read bus. The rate of instruction issue is increased through use of

buffering in instruction control. The buffer is a high density logic (HDL) storage instruction stack which holds four swords of instructions arranged in 16 addresses of 128 bits (quarter-sword) each (figure 23). Each request to central storage transfers one sword of instructions into the instruction stack. This sword of instructions arrives in the stack at a rate of one quarter-sword each minor cycle. The read next sword (RNS) lookahead mechanism makes a request for the next sword of instructions when instructions issue from the most recently acquired sword of instructions. The program may branch forward in the instruction stack to any location in the same sword of instructions (or to the next sword after it is loaded into the stack). It may branch back in the stack to any executed instruction remaining in the stack which was loaded after the last branch out of the stack. The instruction stack is effectively cleared upon branching out of the stack.

Each sword of instructions obtained from central storage is accompanied by 16 parity bits. The hardware checks parity on each 32 bits of instruction at the time the instruction is read out of the instruction stack. A parity error will stop the CPU prior to execution of that instruction.

## (2) Operand Control

The stream unit interfaces with the SAC, floating point pipe 1 and floating point pipe 2.

In scalar operations, many of the operand processing units may be performing their specific tasks at the same time. Each may be involved with a different instruction. In vector operations, the activities of the operand processing units complement each other. Conflicts between operand processing units in vector operations do not normally occur because of the order that the microcode imposes on the processing units. The effect of conflicts between processing units, or of conflicts within a single processing unit, is felt primarily by the scalar, register-to-register, instructions. Conflicts between processing units arise primarily from the storing of results into the register file. Instruction control obtains operands during the issue time, and the operands are issued to an appropriate processing unit at the end of this period of time. At a certain time after issue, a result operand is available at the output of the processing unit; and a data path must be available to transfer the result to the register file. Operands are not issued to a processing unit until it has been ascertained that the result will not arrive at the register file in the same cycle as the result from a previous instruction.

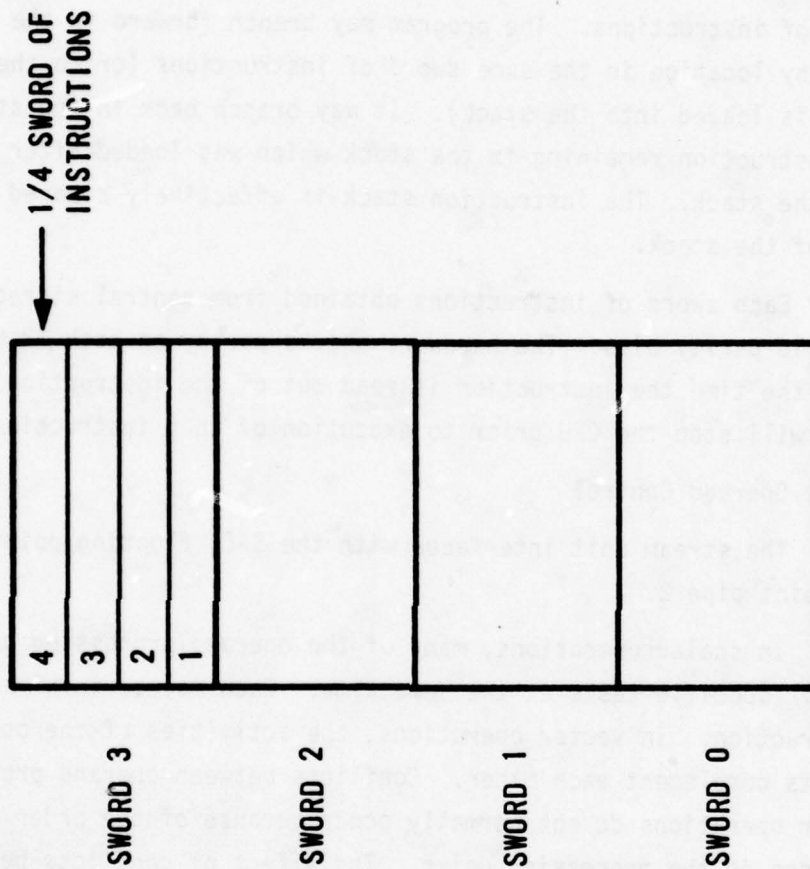


Figure 23. STAR-100 Instruction Control Stack

The pipeline characteristics of each processing unit are such that it is unnecessary to wait for a result before entering the next pair of operands. In other words, operands may be presented to the unit at a rate greater than one pair per unit propagation time. It is possible, for example, to have pipe 1 processing an add and a multiply at the same time. This pipeline capability is not without restriction. Although pipe 2 is used by vector and string instructions in a true pipeline fashion, control of the pipe for scalar instructions is restricted in such a way that only one pair of operands can be processed at a time. As an example, two sequential square root operations in pipe 2 would take almost twice as long as one. By contrast, two sequential multiply operations in pipe 1 require only two more minor cycles than would one.

### (3) Addressing

The computer system uses two modes of addressing central storage:

Virtual addressing

Absolute addressing

Virtual addressing provides an efficient, dynamic method of allotting portions of central storage to each job program by the monitor program. Virtual addressing is used exclusively when the CPU is in the job mode. The switching of the CPU to the monitor mode automatically disables virtual addressing. However, central storage recognizes all addresses as being absolute. Thus, the virtual addressing control circuits convert virtual addresses to the corresponding absolute addresses.

The absolute address is the combination of the absolute page address from the associative word in the page table and the word, half-word, byte, and bit identifier portions of the virtual address. However, before an absolute address is formed, the virtual page address portions of the virtual address and an associative word from the page table must be equal.

A detailed description of addressing operations and formats is given in appendix H.

### (4) Register File

The stream unit contains a register file composed of two 64-word by 128-bit semiconductor memories. The computer uses the register file for instruction and operand addressing, indexing, field length counts, and as a source or destination for register-type instruction operands and results. The 8-bit

designators, in the instructions, address the register file as 256 64-bit registers or address the first (lower) half of the register file as 256 32-bit registers. The register file is subdivided into six major areas:

Machine Registers

Temporary Registers

Global Registers

Environment Registers

Register Save Area

Parameter Registers

A functional description of the registers can be found in appendix I.

#### (5) Operand Shift Network

The operand shift network performs the final pairing of the operands before they enter the floating point pipes. The A and B stream buses (128 bits wide) enter the operand shift network from either the register file or the stream input network. The operand shift network is capable of any shifting on 32-bit boundaries. After pairing, the operands are sent to the floating point pipes via two 64-bit trunks to each pipe.

This network also contains circuits which may select either the A stream, B stream, upper register file, or lower register file for transmission to the data interchange.

#### (6) Microcode Memory

A semiconductor microcode memory of 1536 (224-bit) words is used as part of stream control. The control signals and enable conditions produced by the microcode are used together with hardwired control to manage certain instructions and to process interrupts. During system operation, the microcode memory is used exclusively as a read only memory. The memory must be loaded from the MCU, which also is used to read the microcode status and set certain conditions.

#### c. String Unit

The string unit processes strings of decimal and binary numbers input through the X stream and Y stream. Processing can take the form of character

editing which is performed by the Edit Control hardware and logical operations which are performed by the Logical Instruction Control hardware. This logical control performs the exclusive OR, AND, inclusive OR, and other instructions on input data fields. The string unit also performs move, compare, merge, pack, and unpack operations.

#### (1) Binary Arithmetic Control

This control performs the binary add, subtract, multiply, and divide operations on operand strings. The add, subtract, and divide operations are executed in one 16-bit adder. The multiply operation uses four consecutive 16-bit half adders and 20-bit full adder to generate partial products. The partial product from one pass is added to the partial product of the previous pass in the 16-bit adder used for binary add, subtract, and divide.

#### (2) Decimal Arithmetic Control

This control performs the decimal add, subtract, multiply and divide operations through the use of two 16-bit decimal adders, a divide table, and a 4-digit multiply table. The add and subtract operations are performed in the second adder, which also combines the partial results of the successive passes on multiply and divide operations.

#### d. Floating Point Unit

Calculations are performed in the STAR-100 computer using two's complement arithmetic. Floating point numbers in the computer are two lengths, 32 bits and 64 bits (figure 24). The 32-bit format has an 8-bit exponent and a 24-bit coefficient. The 64-bit format has a 16-bit exponent and a 48-bit coefficient. The leftmost bit of each exponent and coefficient is the sign bit.

The floating point arithmetic hardware is divided into two units or pipes. Pipe 1 (figure 25) performs register add, register subtract, register multiply, and all vector arithmetic instructions except divide and square root. Pipe 2 (figure 26) performs register divide, register square root, and all vector instructions. This organization of hardware allows optimum performance for both register and vector divide operations. For vector operations common to both pipe 1 and pipe 2, the data are divided in half with every odd pair of 64-bit operands going to pipe 2 (that is, first pair, third pair, etc.) and every even pair (that is, second pair, fourth pair, etc.) to pipe 1. In 32 bit mode, each pipe divides in half to become two 32-bit pipes. Therefore, two pairs of operands go alternately to each pipe.

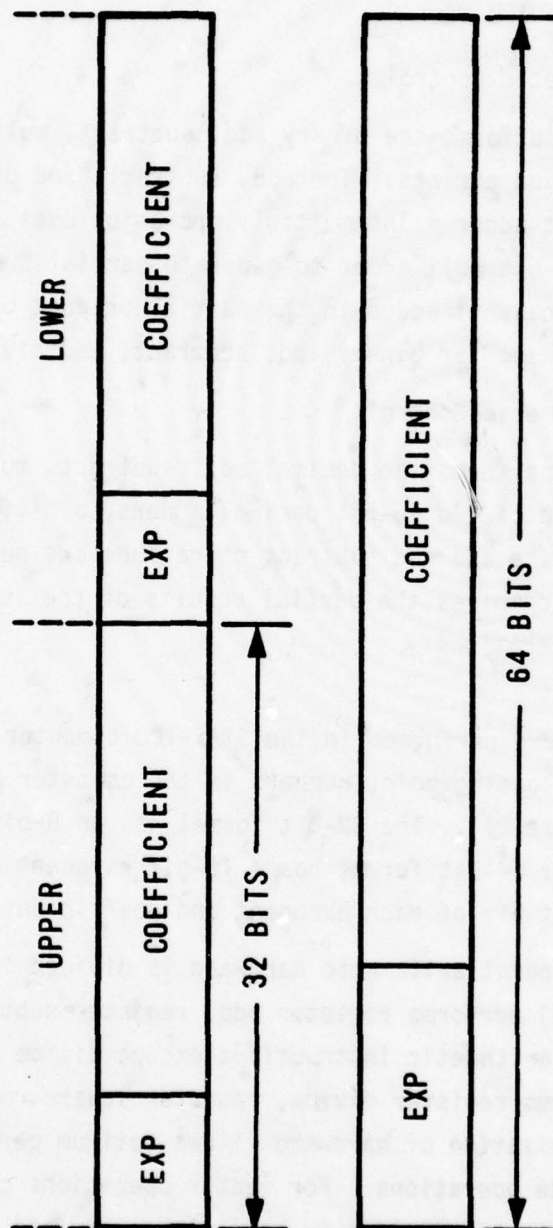


Figure 24. STAR-100 Floating Point Format

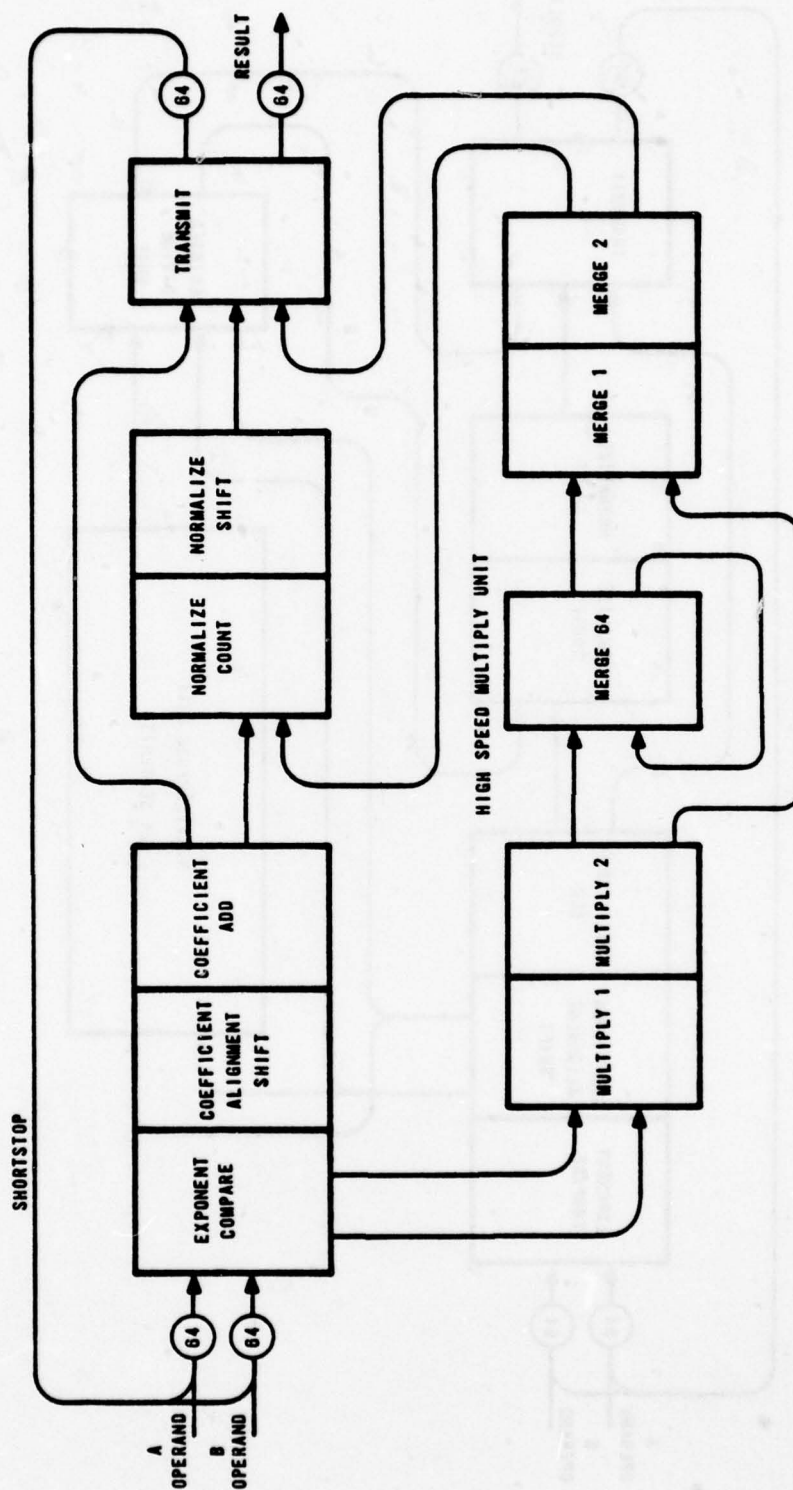


Figure 25. STAR-100 Floating Point Pipe 1

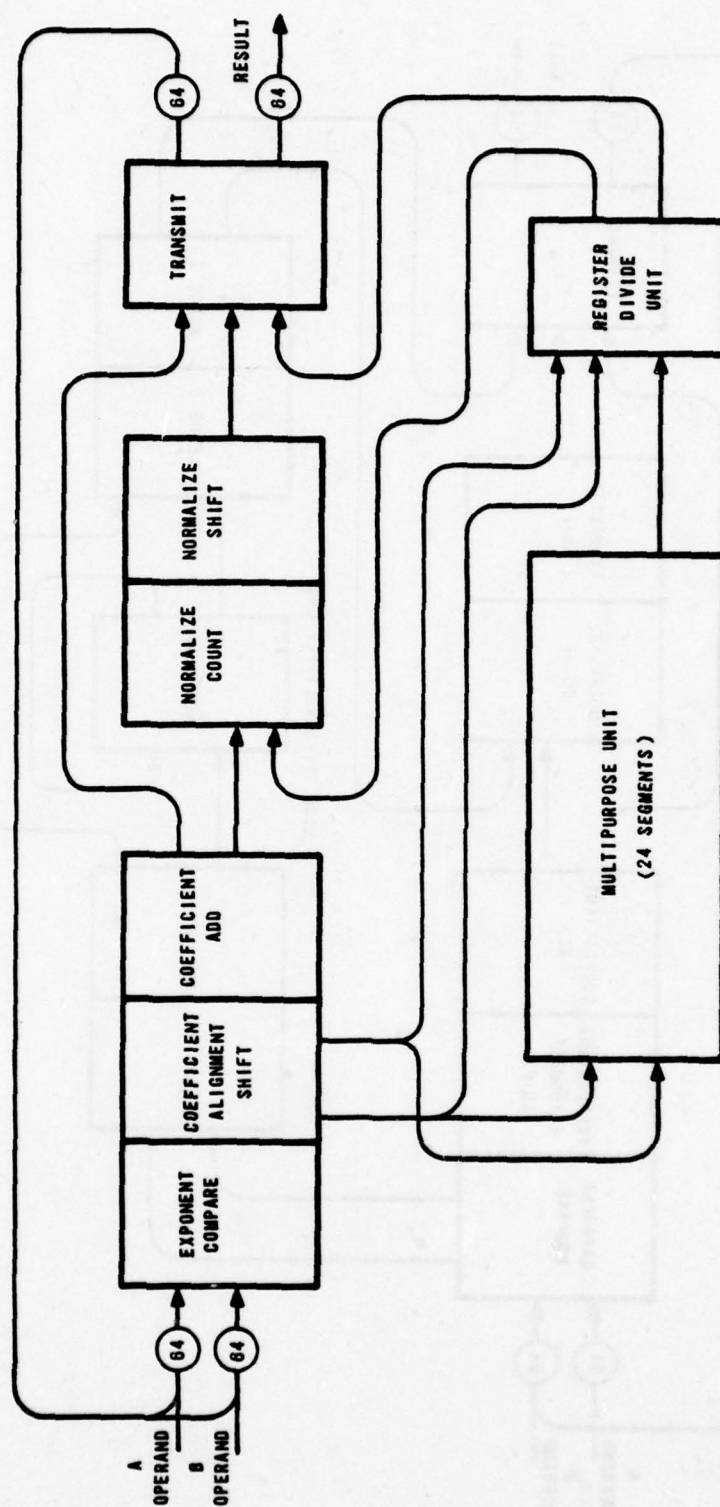


Figure 26. STAR-100 Floating Point Pipe 2

## (1) Pipe 1

Floating point pipe 1 receives operands from the stream unit, performs the instructed operation, and returns the results to the stream unit. Pipe 1 performs arithmetic operations on operands in floating point format and address operations on nonfloating point numbers. Arithmetic operations include such operations as add, subtract, multiply, truncate, adjust exponent, contract, extend, and compare. Address-type operations are those which manipulate various parts of instructions and registers for addressing and indexing purposes. These include operations where the rightmost 16 bits of the instruction transfer to the leftmost 16 bits of register R. The rightmost 48 bits of register R remain unchanged.

For addition and subtraction operations, the input exponents are compared in the exponent compare circuit. The difference in the two exponents is used as a shift count. This shift count determines the amount the coefficient with the smaller exponent is right shifted in the coefficient alignment section. The coefficients are added in the add section. If the operation being performed specifies normalization, the result of the add operation is fed to the normalize count. This circuit produces a shift count which controls the normalize shift network and modifies the result exponent. The transmit circuit returns the shifted result to the stream unit.

If normalization is not specified, the result of the add operation is the desired result and is transmitted to stream.

If the instruction is a multiply, the operands are multiplied in the high-speed multiply unit. The result of the multiply is either returned directly to the transmit section or to the normalize count logic for normalization. The normalize count functions only for the multiply significant instructions.

Any result from pipe 1 may be returned directly to either of the inputs of pipe 1 if the result is needed as an input operand. This process is called shortstopping and eliminates the time necessary to store the result in the register file and then to read it out.

## (2) Pipe 2

Floating point pipe 2 (figure 26) receives operands from the stream, unit, performs the instructed operation, and returns the results to the stream unit. Pipe 2 performs only two address type operations. These are the vector add and subtract address instructions. Pipe 1 and pipe 2 are similar except pipe 2 has a high-speed register divide and a multipurpose unit.

Within pipe 2 the register divide unit performs all register divide operations and binary to binary coded decimal (BCD) and BCD to binary conversions. This is a single segment unit and the operands loop within the unit until the result is reached.

The multipurpose unit performs the square root, vector divide, and vector multiply instructions. The multipurpose unit contains 24 segments. Each segment performs an add type operation. The segments are arranged in four groups of six segments per group. In 64-bit mode, the operands loop on each group, going through each group twice. In 32-bit mode, the operands proceed from segment to segment going through all of them only once. The multipurpose unit delivers its results to the normalize or transmit portions of pipe 2.

#### 4. INSTRUCTIONS

The STAR-100 Computer has 10 types of instructions grouped according to the operations they perform. These instruction types are as follows:

Index - indexing using full and half-words

Register - full and half-word floating point  
full and half-word normalized  
add, subtract, multiply, divide,  
square root, truncate, and shift

Branch - logical branch tests on full and half-word arguments.

Vector - upper, lower, and normalized  
add, subtract, multiply, divide  
truncate and absolute value

Sparse Vector - upper, lower and normalized  
add, subtract, multiply, and divide

Vector Macro - e.g., average, dot product, polynomial evaluation  
scalar-vector product, sum

String - binary and decimal  
add, subtract, multiply and divide

Logical String - AND, AND not, inclusive and exclusive OR,  
NAND, NOR, OR not

Monitor - idle, load, and store associative registers

Nontypical - load, compress, vector compare, etc

A more detailed description of these instruction types is given in appendix J.

## a. Instruction Format

The 32-bit and 64-bit instruction words have 12 types of formats. The formats have hexadecimal numbers, 1 through C. The bits in the instruction word formats number from left to right, 0 through 31 or 0 through 63. Each instruction word format is divided into bit groups that have assigned instruction designators.

## b. Instruction Timing

Table 6 shows some basic information on the execution times for a small subset of STAR-100 instructions. The instruction times given here are for 64-bit normalized register to register operations for both scalar and vector operations. As a basis for understanding, we have assumed that the hardware is running at optimal speed with no register, memory, or functional unit conflicts. The execution time for scalar instruction is the sum of the issue time and the result-to-register file time.

Table 6  
STAR-100 TYPICAL INSTRUCTION TIMES (CYCLES)<sup>a</sup>

<u>Operation</u>	<u>Scalar</u>	Vector
		<u>Set-up Time/Execution Rate</u> <u>(cycle/result)</u>
Floating Add	11	$71 + 4 (N/8)/0.5$
Floating Subtract	11	$71 + 4 (N/8)/0.5$
Floating Multiply	15	$159 + 8 (N/8)/1$
Floating Divide	43	$167 + 16 (N/8)/2$
Floating SQRT	71	$155 + 16 (N/8)/2$

<sup>a</sup> 1 cycle = 40 ns

Issue time is the period of time for which instruction control gathers the required input operands and, if required, ensures that there is a place to put the result operand for one particular instruction. At the end of this period of time, the operands and any necessary control information are issued to an operand processing unit. After issue, the selected operand processing unit generates the actual, functional relationship required between the input operands and when completed, sends the result operands to the desired destination. At the same time, instruction control begins acquiring operands for the next instruction.

The result-to-register file time is the number of cycles after issue that elapses before the result operand produced by the instruction has been returned to the register file and is available for use by a subsequent instruction. Instructions that must use an operand produced by a preceding instruction and that must obtain this operand from the register file cannot be issued before the result-to-register file time for the instruction producing the operand has elapsed.

Execution time, as it applies to vector instructions, is the time period when instruction control begins to obtain operands and to prepare the necessary hardware sections for processing information until the time that the operation has progressed far enough to permit instruction control to begin acting on the next instruction, i.e., the time from the issue of the previous instruction to the issue of the current instruction. This time includes all of the operand processing time, but it does not necessarily include the time required to return the last operand to its destination, usually in the memory.

## APPENDIX A

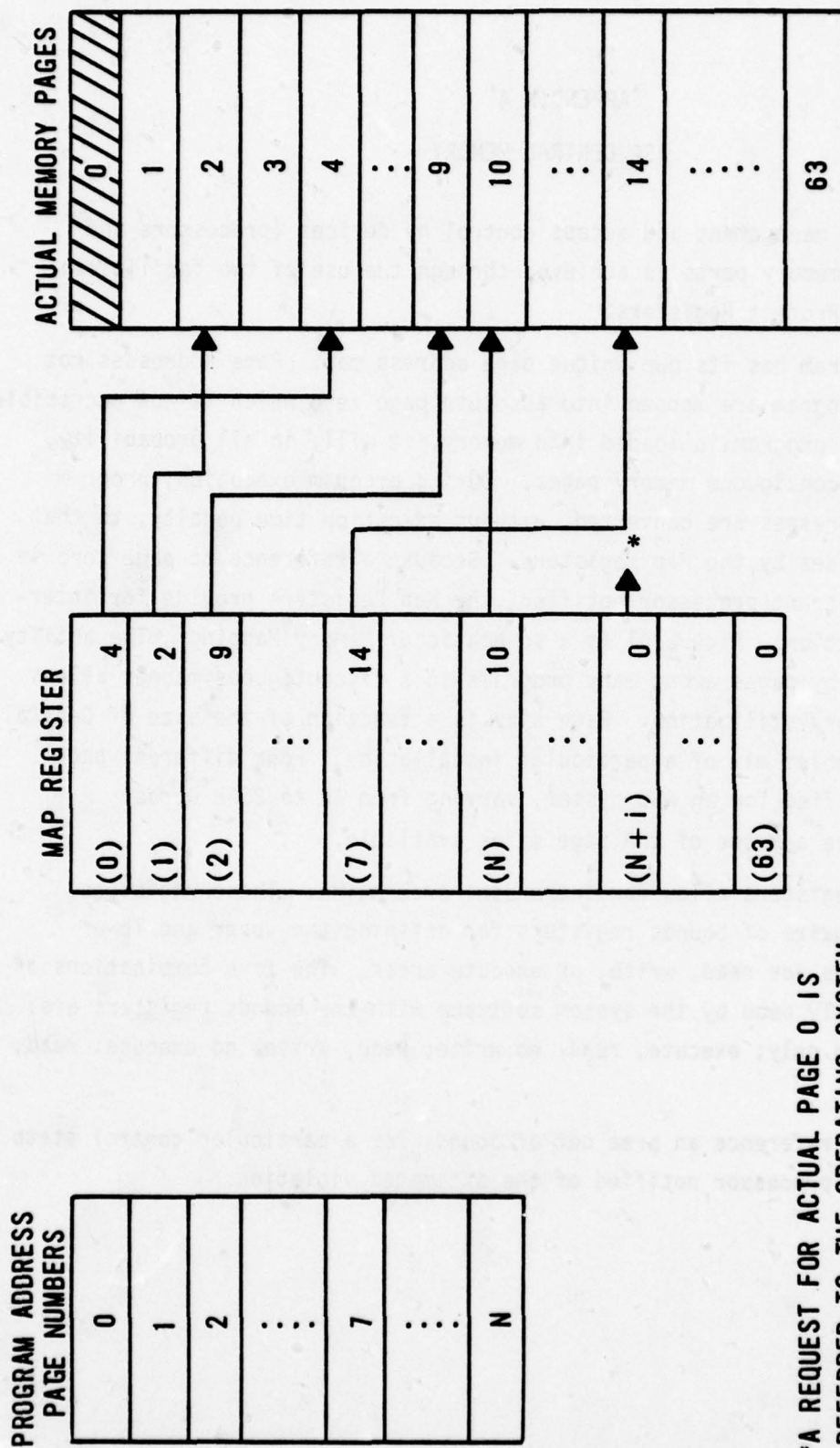
### ASC/CENTRAL MEMORY

Central Memory management and access control by devices (processors and control units) on memory ports is achieved through the use of two facilities: Map Registers and Protect Registers.

Each user program has its own unique page address map. Page addresses not required by the program are mapped into absolute page zero which is not accessible to the CP. When a program is loaded into memory, it will, in all probability, be loaded into discontinuous memory pages. During program execution, program developed page addresses are converted, without execution time penalty, to the actual page addresses by the Map Registers. Because a reference to page zero is denied and the relevant processor notified, the Map Registers provide for inter-user memory protection. Figure A1 is a schematic of Memory Mapping. The ability to segment memory by pages among many programs in a discontinuous manner allows for efficient memory utilization. Page size is a function of the size of Central Memory and the problem mix of a particular installation. Four different page sizes may be specified for an ASC system, varying from 4K to 256K words. A program may utilize any one of the page sizes available.

The Protect Registers allow for intra-user protection. These registers consist of three pairs of bounds registers for defining the upper and lower addresses of access for read, write, or execute areas. The five combinations of protection presently used by the system software with the bounds registers are: execute only; read only; execute, read, no write; read, write, no execute; read, write, execute.

An attempt to reference an area out of bounds for a particular control state is denied and the processor notified of the attempted violation.



\*A REQUEST FOR ACTUAL PAGE 0 IS REFERRED TO THE OPERATING SYSTEM FOR RESOLUTION (INVALID REQUEST, ERROR, ETC.)

Figure A1. ASC Memory Mapping

## APPENDIX B

## ASC/PERIPHERAL PROCESSOR

## 1. SINGLE WORD BUFFER (SWB)

The SWB provides VP access to CM. The SWB consists of eight 32-bit data registers, eight 24-bit address registers, and controls. Each of the eight register sets has a fixed association with one of the eight VPs. Viewed by a single VP, the SWB appears to be a memory data register and a memory address register.

At any given time the SWB may contain up to eight memory requests, one for each VP. These requests are processed on a combinational priority/first in, first out basis. There are two priority levels; and if two or more requests of equal priority are unprocessed at any time, they are handled first in, first out.

When a request arrives at the SWB, it automatically has a priority assignment determined by the CM priority file maintained in one of the CRs. The file is arranged by VP number, and all requests from a particular VP receive the priority encoded in 2 bits of the priority file. The contents of the file are programmed by the monitor, and the priority code assignment for each VP is a function of the program to be executed by the VP.

## 2. READ-ONLY MEMORY (ROM)

The ROM contains a pool of systems programs and is not accessed except by reference from the program counter. The pool includes a skeletal monitor program and at least one control program for each I/O device connected to the system. The ROM has an access time of 25 ns and provides 32-bit instructions to the VP units. Total program space in ROM is expandable to 4K words. The memory is organized into 256 word modules so that portions of programs can be modified without complete refabrication of the memory.

The I/O device programs can include control functions for the storage media as well as data transfer functions. Thus, motion of mechanical devices can be controlled directly by the program rather than by highly special-purpose hardware for each device type. Variations of a basic program are provided by parameters supplied by the monitor. Such parameters are carried in CM or in the accumulator registers of the VP executing the program.

### 3. VIRTUAL PROCESSORS (VP)

The eight VPs share PP elements. To implement this sharing, time is divided into cycles with each cycle containing 16 time slots. Each time slot is one bit period (85 ns) in duration. These time slots are assigned to VPs on the basis of time slot availability and program requirements. A VP is operative whenever a time slot assigned to that VP occurs. Thus, if a VP has been assigned no time slots, it is not executing a program. If a VP has been assigned one of the 16 time slots, it is executing a program 1/16 of the time. More than one time slot can be assigned to a single VP. The monitor VP turns the slave VPs on and off by manipulation of the time slot assignments.

The major components of each VP include: program counter (PC); next instruction register (NIR); instruction register (IR); and four accumulator registers addressable to the byte level. Associated with the IR in each VP is a three-state controller, used to provide a broad definition of the execution state of a given instruction, and a bit counter which provides a finer definition of the instruction step within the state class. When the time slot assigned to a particular VP occurs, the IR and the associated state class controller and bit counter provide control of the shared portions of the PP.

The source of PP instructions may be either ROM or CM. The memory being addressed from the PC is controlled by the addressing mode, which can be modified by the branch instructions or by clearing the system. Each VP is placed in the ROM mode and each PC points to location 0 when the system is cleared.

When the program sequence is obtained from CM, it is acquired via the SWB. Since this is the same buffer used for data transfers to or from CM, and since CM access is slower than ROM access, execution time is more favorable when the program is obtained from ROM.

### 4. COMMUNICATION REGISTERS (CR)

The PP includes up to 64 CRs each of which contains 32 cells. Each CR cell has two sets of inputs. One set is connected into the PP, and the other set is available for use by the peripheral device. Each CR is addressable from the VPs and by the device to which it connects. The CRs provide the control and data links to all peripheral equipment including the system console. Some parameters which control system functioning are also stored in the CRs from where the control is exercised. These assignments are unique to a particular ASC system.

The contents of the first 16 CRs can be protected from modification by individual VPs. The protection is controlled by software via CR bits assigned specifically for that purpose. The VP selected by the VP SELECT switch on the maintenance panel is insensitive to the CR protection scheme, and can always modify the contents of protected CRs.

## APPENDIX C

### ASC/ARITHMETIC UNIT

The Arithmetic Unit (AU) is comprised of seven sections plus the AU receiver register. The following paragraphs describe each of these seven sections.

#### 1. EXPONENT SUBTRACT

The exponent subtract section is primarily responsible for determining the proper inputs to the add section for use in floating add instructions. It is used for both scalar and vector operations and is also responsible for supplying proper input to the accumulator section for floating vector-dot product instructions.

This section determines the difference in the exponents of floating point operands or in the case of equal exponents, which mantissa is larger. Upon determining the larger number, the true or complement of the operands is gated into registers according to the operation to be performed such as Add, Subtract, Add Magnitude, etc. Also, at this time a 7 bit subtracter determines the exponent difference which is used in the align section to properly align the floating point operands.

Since logic is required in this section to determine relative magnitude of the mantissa, the test instructions for greater than, less than, or equal to are also performed in this section to avoid repetition of hardware.

#### 2. ALIGN

The align section is in operation for all floating add instructions or for any right shift instruction. Floating point instructions are performed after one pass through the align section while fixed point shifts require two cycles.

The shift logic has provisions to allow any shift length which is a multiple of four to be performed in one cycle. Since floating point numbers are represented in hexadecimal digits, this will facilitate the very fast floating point additions. The length of right shift can be obtained from the instruction word for a shift instruction or from the exponent difference information as supplied by the exponent subtract section.

Fixed point right shifts are performed by first shifting in one cycle the largest multiple of 4 bits contained within the shift value. Then, the residue of 0, 1, 2, or 3 bits is shifted on the next cycle. This results in a minimum of shift paths into each latch since the 4-bit paths already exist.

### 3. ADD

The add section is shared for many instructions depending only upon which paths are selected into the section. Floating add instructions are entered by way of the align section. Fixed point add operands enter the arithmetic unit at the add section.

The adder is 64 bits in length and contains second level look-ahead logic. The floating point numbers are in the proper format when entering the add section; however, the fixed point operands are modified to reflect either add, subtract, or add magnitude type instructions.

### 4. NORMALIZE

The normalize section is employed for both floating add instructions and fixed point left shift instructions. Divisions are routed through this section to guarantee bit normalized inputs for divide instructions.

This section closely resembles the align section in that floating point operations require only one cycle while fixed point shifts require two. The major difference in the two sections is that the align section is given the information concerning length of shift for hexadecimal digits in floating point. The normalize section has to compute the length of shift required to normalize a floating point number by examining to determine which 4-bit group contains the most significant logical one. An adder is also required to update the exponent when a normalization takes place.

Fixed point left shift instructions are supplied with the length of left shift from the instruction word.

### 5. MULTIPLY

The multiply section is required to operate on both floating and fixed point operands. The floating point numbers are represented in sign and magnitude, the fixed point numbers are in two's complement form. The method of multiplying is based on two's complement operands with the floating point multiplication performed by arbitrarily assigning positive signs during multiplication and then applying the proper sign when multiplication is complete.

The multiplier is capable of multiplying any two numbers up to 32 bits in only one pass through the multiply section. The result is in the form of a pseudo-sum and carry which must be added to obtain the result. The sum and carry are added in the accumulator section.

The multiply section is also used to perform division by a sequence of multiplication operations.

## 6. ACCUMULATOR

The accumulator totals the pseudo-sum and pseudo-carry from the multiplier section to form the product of a multiplication. This section also performs running totals for vector operations such as a Vector-Dot-Product instruction.

Like the add section which was previously described, the accumulator performs a second-level look-ahead to facilitate a fast addition.

## 7. OUTPUT

All results to be sent to the CP must be gated through the output section. Information could have originated in any one of the other sections of the AU with the exception of the multiply section.

Simple instructions such as Booleans, transfers, masks, etc., are performed in this section and gated out in one pass through the section.

## APPENDIX D

### ASC/INSTRUCTION TIMING

#### 1. SCALAR INSTRUCTIONS

Scalar instruction processing time in the Central Processor (CP) can be predicted with a fair degree of accuracy by considering the following factors:

- Arithmetic Unit clock time
- Operand fetching time
- Register conflict delay
- Multiple store instruction delay
- Read after write delay
- Indirect address generation time
- Execute instruction fetching time
- Instruction fetching time after a branch instruction without look ahead
- Instruction hazard refetch time
- Short Circuit Path

##### a. Arithmetic Unit (AU) Clock Times

The AU clock times are defined as the number of clock periods required to propagate input arguments through the AU, to the AU output registers. Certain instructions can be executed in sequence by the AU without creating periods of inactivity or delays in pipeline flow.

##### b. Operand Fetching Time

Operand fetching time refers to CM access time for obtaining memory operands. This time is measured from the clock period at which the effective address of the operand is at the address output register of the IPU to the clock period at which the requested operand resides in the output register of the MBU. This time is normally 10 periods if there are no memory conflicts at the MCU or priority delays at the MBU memory controller.

The X and Y buffer registers, used for streaming vector operands into the AU, can be used during scalar operations to retain the most recently used operand octets from CM. If a request for a word in CM is not contained in either the X or Y buffers, the octet of words containing the requested word replaces the

contents of the X-buffer if Y was last used or replaces the contents of the Y-buffer if X was last used. An effective address request for a word in an octet which is presently contained in either the X or Y buffers is terminated at the buffer (the address is not sent to CM) and the intended operand is read from the buffer in which it is resident. There is no pipe delay when the required operand is resident in either the X or Y buffer registers.

If two successive instructions request CM operands from different octets and neither one is resident in the X or Y buffers, then it is possible for both requests to be issued to CM before the IPU needs to be stopped to wait for CM access. This provides overlap of CM requests, rather than having to wait the entire memory cycle time for each memory octet fetch. The second octet request can be placed on the CM address bus two cycles after the first octet request. The second octet of data will be available in the second buffer two cycles after the first arrives providing that no memory conflicts occurred and that the second read was from an alternate memory module than the first. If the two read requests were to the same stack, then an additional two cycles will elapse before the second read data is available at the buffer register due to memory stack conflict.

An instruction can proceed to the AU without memory delay if the required operand is presently residing in either the X or Y buffer registers.

c. Register Conflict Delay (RCD)

An RCD occurs whenever an instruction requires the contents of a register (base, index, general arithmetic, or vector parameter) and that register is presently in the process of being modified by a previous instruction which has not yet passed through the AU output level. Register conflicts occur because of the pipeline nature of the CP. An RCD can occur at any of the first three levels of the IPU; the Instruction Register (IR) level 1, the Preindex (XR) level 2, or the Index (AR) level 3.

d. Multiple Store Instruction Delay

A multiple store instruction delay is caused when two or more store instructions, all with different octet addresses, occur consecutively or with only one instruction separation in an instruction stream. The MBU and AU pipe sections are provided with one address register to retain the octet address of one store instruction. A second store instruction occurring in a rapid sequence will be delayed at level 3 of the IPU until the first store instruction of the sequence has passed to the AU output level.

A delay due to CM write conflicts may also occur for any two or more store instructions which are too closely spaced, but that is a different type of delay from the multiple store instruction delay being considered here. The multiple store delay has a tendency to ease the memory write conflict problem, since the pipeline operates at a reduced speed when the multiple stores are detected.

There is no such multiple store delay for a series of two or more consecutive store instructions which all address the same octet or which write consecutively into monotonically increasing or decreasing address locations.

e. Read After Write Delay (Same Octet)

This delay is caused by attempting to read from a CM location while a previous write instruction is still in the process of writing into the same location or into the same memory octet. A write instruction is in the process of writing into memory if it is anywhere below the IPU, but not yet in CM.

It is possible to acquire a modified operand over the Z and X update path providing no other stores into different memory octets exist between the store instruction whose octet address agrees with the operand read octet address of the instruction presently at level 3 of the IPU. The update from Z to X occurs after all stores into the agreeing octet (which are in either the MBU or AU) have been entered into the Z-buffer. The update may occur simultaneously with the arrival of the read data from CM. In this case, the read data must be merged with the update information from Z.

f. Indirect Address Generation Time

Each level of indirect addressing requires 10 clock periods assuming no memory conflicts.

g. Execute Instruction Fetching

Each Execute instruction fetch requires 10 clock periods assuming no memory conflicts. The only difference between Executes and Indirects is that an Execute instruction is fetching an instruction to be executed whereas an Indirect reference is fetching the address of an operand or the address of the address of an operand, etc.

h. Instruction Fetching After Branching

A delay equivalent to that of Indirect or Execute occurs when fetching instructions following a branch without look-ahead.

#### i. Instruction Hazard Condition

An instruction hazard condition exists when a store instruction is in the process of modifying a word in a CM octet from which instructions previously read are currently residing in the IPU. These instructions are then the "old" commands and hence, the CP must prevent their being executed.

Prevention of execution is accomplished simply by cancelling the instructions above the point in the IPU where instruction addresses agree with store addresses. In order to restart the program, the CP must wait until the culprit store instruction has completed writing into CM, then the IPU must recall the instructions which were cancelled.

#### j. Short Circuit (SC) Path

An SC path exists from the AU output register to the AU receiver register. This path is used whenever an instruction in an instruction stream requires the same register of the CP register file as the immediately preceding instruction. In this instance the preceding instruction must be one which will store into the same register that the succeeding instruction requires as its register operand. When this condition arises, the succeeding instruction will not wait for the normal register conflict delay, but instead will proceed down the CP pipeline without its register operand and will acquire the operand via the AU SC path upon the arrival of the succeeding instruction at the AU.

Short circuit conditions can only occur for adjacent pairs of instructions of the same word size (double with double, single with single, etc.). Short circuit can occur for an unlimited number of instructions in a chain as long as they all use the same register and have the same word size. Any single instruction which does not use the same register will break the chain. Also, two successive branch instructions which use the branch test level to determine the outcome of the branch condition cannot use the SC path to achieve a faster branch decision for the second of the two branches because the branch test level does not have a path equivalent to the AU SC path. The branch test level does not solve for the value of the argument in an Increment Test and Branch Instruction but rather determines only whether the branch test passes or fails.

The timing for the SC path can be determined by following the first of the pair of instructions down the pipeline to the AU output level. The time at which the result of the first instruction arrives at the AU output can be

determined. The second instruction will advance to the MBU output level and wait there, if it has arrived before the first instruction's result is available from the AU output. One cycle is used to route the AU result back to the AU receiver level. The second instruction advances through the AU when all of its required arguments are supplied at the AU input.

## 2. VECTOR INSTRUCTIONS

Vector priming is divided into five distinct processes: Vector parameter file fetch; MBU initialization; Memory operand fetch; AU fill; and AU empty.

When a vector instruction terminates, a process of AU emptying occurs. This process involves only the depletion of operand arguments from the AU.

### a. Vector Parameter File (VPF) Fetch

The VPF fetching occurs as a result of specifying a vector instruction for which a new VPF is requested from CM. The new VPF is requested when the effective address of the vector instruction has been developed by the normal IPU indexing hardware. This hardware does the preindexing and index addition required for generating the effective address.

The VPF fetching begins after the vector instruction has reached the index addition level (level 3) and the VPF address has been developed. The VPF fetching requires 8 clock periods. However, this CM request is overlapped in time with the previous scalar instructions presently being processed downstream by the AU. Loading of a new VPF appears no different to the IPU than if the IPU had encountered a scalar Load Register File instruction. The fetching of data for these files is carried out entirely by the IPU, and the MBU has not been involved until now with the vector instruction. Also, the memory fetching time for the VPF will be less than the time required for memory operand fetching because the register files have a simpler interface with memory than the interface that exists at the MBU.

Overlapping memory cycles between the IPU and MBU during VPF fetch will exist if the scalar instruction immediately prior to the vector instruction requests a central memory operand from a new octet and all previous scalar memory requests have been granted (data received from memory). If this condition exists, the scalar instruction requesting the new octet is allowed to advance beyond level 3 (index addition level), providing that "path ahead" is clear, and level 3 is filled with the developed address for the VPR request of the vector instruction. Thus, the time required to fetch the VPF is completely overlapped by the time required for fetching an operand of a prior scalar instruction. The VPF fetching is essentially free in this case.

The vector instruction could be one which specifies the use of the vector parameter data currently residing in the VPF registers. In this case, no VPF fetch cycle is required, and the vector priming operation proceeds immediately to the MBU initialization process.

The MBU initialization begins upon detection of a vector instruction at level 4 of the IPU. This MBU initialization is then overlapped with previous scalar instruction processing going on downstream in the AU.

#### b. MBU Initialization

Initialization of the MBU involves the transferring of vector parameter data from the VPF registers in the IPU to the vector working registers of the MBU. This process begins immediately after new data have been entered into the VPF registers, if a VPF fetch is specified, or immediately upon detection of a vector instruction in level 4 of the IPU if a request for the current VPF data is specified.

The MBU initialization requires 10 clock periods. This time begins with the starting address development in the IPU for vectors A, B and C (in that order). Development utilizes the preindex and index addition levels (levels 2 and 3) of the IPU. Then the remaining five words of the vector parameter data are transmitted one word at a time to the MBU for distribution to its working registers that control the vector operation.

No advantage would be gained by transmitting the remaining inner and outer loop increment information to the MBU at a faster rate, since the memory operand fetch operation is overlapped with the transmission of the remaining data. The transmission of VPF data complete seven cycles before the first operand arguments are available as inputs to the AU receiver register, even though the VPF data are sent only one word at a time.

#### c. Memory Operand Fetch

This cycle begins five cycles after MBU initialization starts. It is considered to start at the time the first address reaches the central memory address requestor in the MBU.

Memory operand fetching of the first octet of data for vector A and B is completed when the first two operand arguments are placed in the MBU output registers and are available as inputs to the AU. The process of first operand octet fetch requires 12 clock periods. Subsequent octet fetches are requested

at 8 clock intervals during the self-loop, and the pipeline flow of operands to the AU is maintained throughout the vector operation. The initial operand fetch is an overhead penalty which occurs only once during the vector priming procedure.

d. AU File

The AU can proceed to fill its internal pipe sections as soon as the operand stream is presented to its input registers from the MBU. An AU receiver register accepts the operands from the MBU, which have been transmitted between physical cabinets containing the MBU and the AU. The receiver register, therefore, adds 1 clock period to the AU operation times given for scalar instructions in the timing section for scalars. This figure (scalar AU time plus 1) gives the number of clock intervals before the first result appears at the AU output. Scalar AU times vary from instruction to instruction, but once the AU has been filled in a vector mode, AU results are produced every clock period for most single length vector instructions with a few exceptions.

e. AU Empty

At the termination of a vector instruction, the AU will exhaust the final results into the Z-buffer registers, and a Z-write operation is forced to purge the output buffers of the vector results so that scalar hazard detection can begin fresh.

However, when a scalar instruction follows a vector, overlapping occurs again. Two general constraints need to be listed: (1) If the subsequent scalar instruction uses indirect addressing, it will wait in level 1 until the vector operation is completely terminated. This prevents an erroneous indirect linkage through an area of CM which is being modified by a vector instruction. (2) If the vector instruction is of the class requiring the storage of an item count at the completion of each self-loop, then a subsequent scalar instruction must wait at level 1. Vectors which store item counts use the 2nd and 3rd levels of the IPU to restart the vector in case a context switch operation prematurely terminate the vector.

Overlapping of the subsequent scalar begins after the MBU has determined that all self, inner, and outer loops are completed and that the ZB register has initiated its last write cycle for the vector instruction. At this time the subsequent scalar may use the facilities of the MBU to request a memory operand required for scalar instruction execution. Thus, the requesting of a next scalar octet is overlapped with the termination (AU empty) of the present vector instruction in the AU.

## APPENDIX E

## STAR-100/STORAGE ACCESS CONTROL

## 1. SAC READ OPERATIONS

The SAC unit contains three read accesses (figure 22). An access is defined as a grouping of one or more buses which share a selection network for accessing MCS. Four banks in a memory section share a common read data path. One 132-bit data bus carries data from each MCS section to SAC. This requires four quarter-sword transfers to read one 528-bit sword. The first quarter-sword leaves MCS five minor cycles after the request is received, and the remaining quarter-swords are transmitted on the next three minor cycles.

On read operations, SAC performs an odd parity check on each half-word of data. If a parity fault is detected, the parity fault condition is set. The resulting operation depends on the access input that requested the data containing the parity fault as described in a subsequent subparagraph. If no parity fault is detected, the data are transmitted to the input that made the request. Since instruction words are transmitted over a read bus, the SAC unit first checks the parity in the normal manner and then transmits the 128 data bits with the corresponding parity bits to the stream unit for further checking.

## 2. SAC WRITE OPERATIONS

The SAC unit contains two write accesses (figure 22). Write buses provide two inputs for the stream unit access to MCS. These two write buses transmit result operands and other output data from the stream unit to SAC for storage in MCS. The SAC unit assembles the 16-bit bytes transmitted from the I/O channels into quarter-swords and transmits these to MCS. Four banks in a section share a common write data bus. One 132-bit data bus carries data from SAC to each section; therefore, each sword transfers as four quarter-sword bytes. The first quarter-sword arrives at MCS one minor cycle after the request. The remaining quarter-swords are transmitted on the next three minor cycles. The I/O channels share the other write access with the stream unit. The stream unit uses this write access for exchange operations only. In write operations, the SAC unit generates the four parity bits for each quarter-sword. The format of the write data, as transmitted to MCS, is identical to the read data.

## APPENDIX F

## STAR-100/CENTRAL PROCESSOR MODES

The CPU operates in one of two programming modes: Monitor mode; Job mode.

The CPU automatically exchanges from the job mode to the monitor mode when it receives an interrupt or when a job program executes an exit force instruction. The monitor mode disables all interrupts and virtual addressing and permits absolute addressing to central storage. Any interrupts that occur during the monitor mode temporarily store until the monitor program executes an idle or an exit force instruction. The idle instruction causes the CPU to wait until an interrupt occurs. The exit force instruction switches the CPU to the job mode and starts executing the selected job program. Switching to the job mode enables the interrupts and virtual addressing.

The purpose of the exchange is to change the prime role of the CPU. In job mode, job tasks are performed. In monitor mode, the system decisions are made and the page table is altered.

## APPENDIX G

## STAR-100/I/O ASSEMBLY AND DISASSEMBLY

Each I/O channel contains a 32-bit assembly/disassembly register and address register circuits. In addition, a 32-word-by-128-bit high density logic memory is shared by the I/O channels as the I/O buffer. The I/O buffer is used for assembly, disassembly, and buffer operations. An I/O channel can be allocated a quarter, half, or whole sword in the I/O buffer.

The data trunk between the assembly/disassembly buffer (ADB) and central memory is 128 bits wide. The data trunk between the ADB and the channel assembly/disassembly registers is 32 bits wide. The data trunks between the peripheral stations and the assembly/disassembly registers are 16 bits wide.

In I/O write operations, each 32-bit half-word consists of two successive 16-bit transmissions from the peripheral station. The two 16-bit portions are assembled in the assembly/disassembly register for transmission to the I/O buffer.

The starting address for an I/O read or write operation is sent from the peripheral station as two 16-bit transmissions. The first 16 bits contain the upper or lower 500K MCS selection bit and the high-order 4 bits of the sword address. The second 16 bits contain the low-order 7 sword bits, the 5 bank selection bits, the quarter-sword address, and the half-word address. The 11 sword address and 6 bank address bits are transmitted to the channel address register where they are incremented, as sword boundaries are crossed during central storage references. The quarter-sword address bits are sent to I/O control where they determine the quarter-sword that is loaded into or transmitted from the I/O buffer. The half-word address bits determine the 32-bit half-word that is loaded into or transmitted from the I/O buffer.

## APPENDIX H

## STAR-100/VIRTUAL ADDRESSING MECHANISM

## 1. ASSOCIATIVE WORDS

The associative words contain the information necessary to convert a virtual page address into an absolute address. The monitor program must assemble the associative words into a page table as necessary for a given run. If a page is altered, a job program has performed a write operation on at least one bit in the page defined by the associative word. In the monitor mode, the CPU does not use the associative words in addressing. Thus, alteration or referencing storage by the monitor program is not recorded in the associative words.

## 2. LOCK

A lock is a 12-bit quantity contained in each associative word. The lock associates a page of central storage with a job program or several job programs.

## 3. KEYS

The monitor assigns four 12-bit keys to each job. The keys for a particular job are read from central storage as part of the job. The monitor program transfers the keys to the virtual address key register. After the virtual page address portion of an associative word matches with the corresponding portion of a virtual address, one of the four keys for the job must match the lock in the associative word before the storage reference can take place. If a key matches the lock of an associative word for a particular storage reference, but the operation is disabled by the lockout code for that type of reference, a storage access interrupt takes place. A storage access interrupt causes an exchange to the monitor mode.

## 4. ASSOCIATIVE REGISTERS

The SAC unit contains sixteen 64-bit associative registers (ARs). Each AR contains one associative word. The ARs contain the first 16 associative words in the page table. For example, if the computer system consists of 1,048,576 words of central storage and if only 65K-word pages are selected, the associative words for all 16 pages would be contained in the ARs. In the monitor mode, the

contents of the ARs can be stored into, or loaded from, central storage with the store associative registers or load associative registers instructions, respectively.

## 5. SPACE TABLE

The space table shown in figure H1 consists of the locations in central storage that contain the list of associative words. The space table starts at absolute bit address  $4400_{16}$  (word address  $0110_{16}$ ). The space table extends into central storage until an end of page table code is found in the usage bits of the corresponding associative word. Thus, the space table serves as an extension of the ARs to make up a complete page table.

## 6. PAGE TABLE

The page table contains the complete list of associative words and includes both the ARs and space table. The associative words contained in the page table define the pages currently allotted space in central storage. Figure H1 shows the format of the page table. Note that if the associative words in the ARs are stored in central storage with the store ARs instruction, they are stored in 16 consecutive 64-bit storage locations of absolute bit addresses  $4000_{16}$  through  $43C0_{16}$ .

## 7. VIRTUAL ADDRESS FORMAT

Figure H2 shows the virtual address formats for the 512-word page and 65K word-page, respectively. Note that in the 512-word page, the virtual page identifier consists of 33 bits. In the 65K-word page, on the other hand, the virtual page identifier is contained in 26 bits of the virtual address. This difference results from the number of bits needed to locate the word in the page. In the 65K-word page selection, 16 bits are needed to locate the word in the page, giving a word-address range of  $0000_{16}$  to  $FFFF_{16}$ , which is equivalent to  $65,536_{10}$  storage locations. In the 512-word page, the 9-bit word identifier gives a word-address range of  $000_{16}$  to  $1FF_{16}$  (512 storage locations).

The bit, byte, half-word, and word identifier portions of the virtual address are absolute. Thus, when the virtual page identifier is converted into an absolute page identifier, these portions of the virtual address are substituted directly into the absolute address.

AD-A060 670

AIR FORCE WEAPONS LAB KIRTLAND AFB N MEX  
FOURTH GENERATION COMPUTER ARCHITECTURES.(U)  
JAN 78 G W BREZINA

F/G 9/2

UNCLASSIFIED

AFWL-TR-77-190

SBIE-AD-E200 129

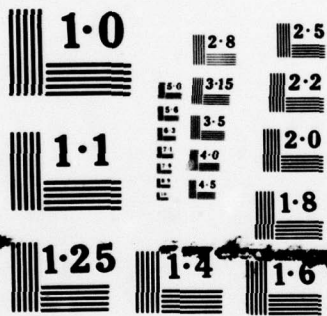
NL

2 OF 2  
ADA  
060670



END  
DATE  
FILMED

1 -79  
DDC



NATIONAL BUREAU OF STANDARDS  
MICROCOPY RESOLUTION TEST CHART

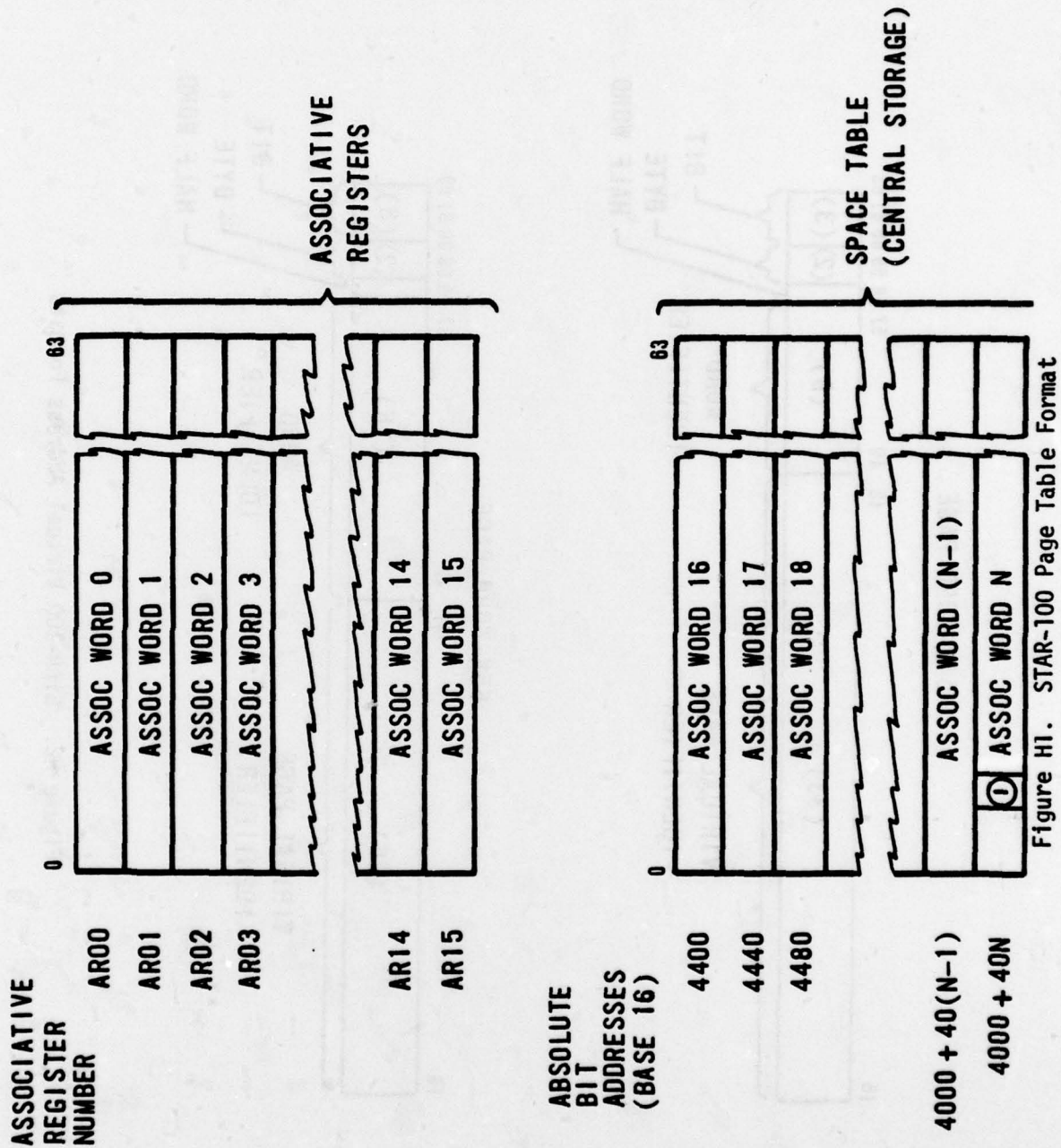
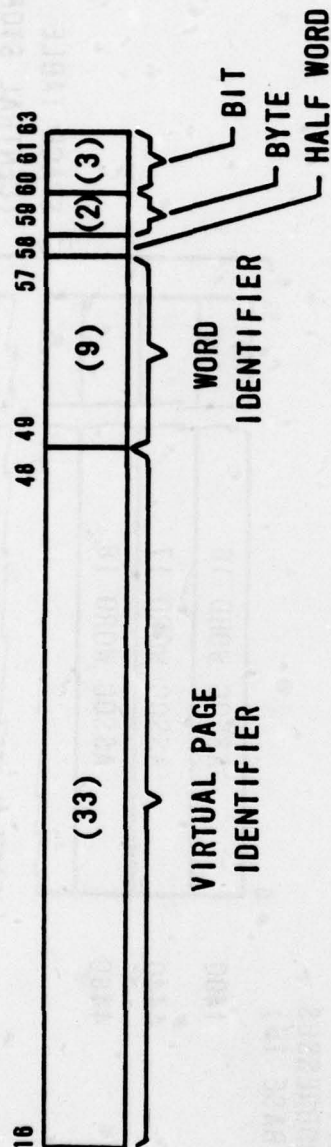


Figure H1. STAR-100 Page Table Format

### 512-WORD PAGE



### 65K-WORD PAGE

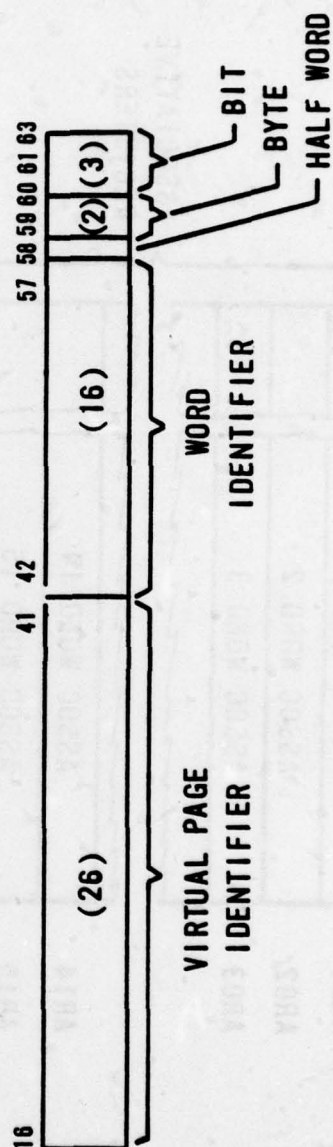


Figure H2. STAR-100 Virtual Address Formats

## 8. OPERATION OF VIRTUAL ADDRESSING

In the processing of a job program, each virtual address is transmitted from the stream unit to the SAC unit. The SAC unit compares the virtual page identifier in the virtual address (figure H2) with the corresponding portion of each associative word in the page table. If the virtual page identifiers match and the lock matches one of the four keys, a match condition occurs. If a match results, the absolute page address associated with the match-producing entry in the page table is combined with the applicable portion of the word identifier sent from stream. The upper 17 bits of this combined address references one sword (eight 64-bit words) from central storage. The remaining word, half-word, byte, and bit identifiers remain in stream and select the word, half-word, byte, and/or bit from the words transmitted from SAC. If the end of the page table is detected with no preceding match condition, or if a match results but the operation is disabled by the lockout code, a storage access interrupt results.

## APPENDIX I

## STAR-100/REGISTER FILE

The register file shown in figure I1 is subdivided into six major areas containing a total of 256 registers.

Machine registers

Temporary registers

Global registers

Environment registers

Register save area

Parameter registers (in pairs)

#### 1. MACHINE REGISTERS

These registers include only registers 0, 1, and 2. Register 0, by convention, contains the number 0. Register 1 contains the data flag branch exit address, and register 2 contains the data flag branch entry address. In monitor mode, additional registers are reserved as machine registers.

#### 2. TEMPORARY REGISTERS

A user program may utilize two areas for temporary storage, addresses, or data. The two areas are from registers 3 to 13<sub>16</sub>, and from 20 to the beginning of the parameter registers.

The lower area is large enough for execution of short subroutines (such as SIN, COS, etc.) completely within the temporary space, obviating the need for saving and restoring any of the caller's permanent registers when short modules are needed by a program. The upper area, which is large enough to hold a variety of user procedures, is never saved by the callee.

#### 3. GLOBAL REGISTERS

The contents of the global registers are universal to all programs within a specific execution/language system. The contents can be assumed by all modules within a given system and are not usually loaded, saved, or restored by called modules. The values in these registers are unique to a given operating environment; thus, if a module from a different environment is to be called, it is the

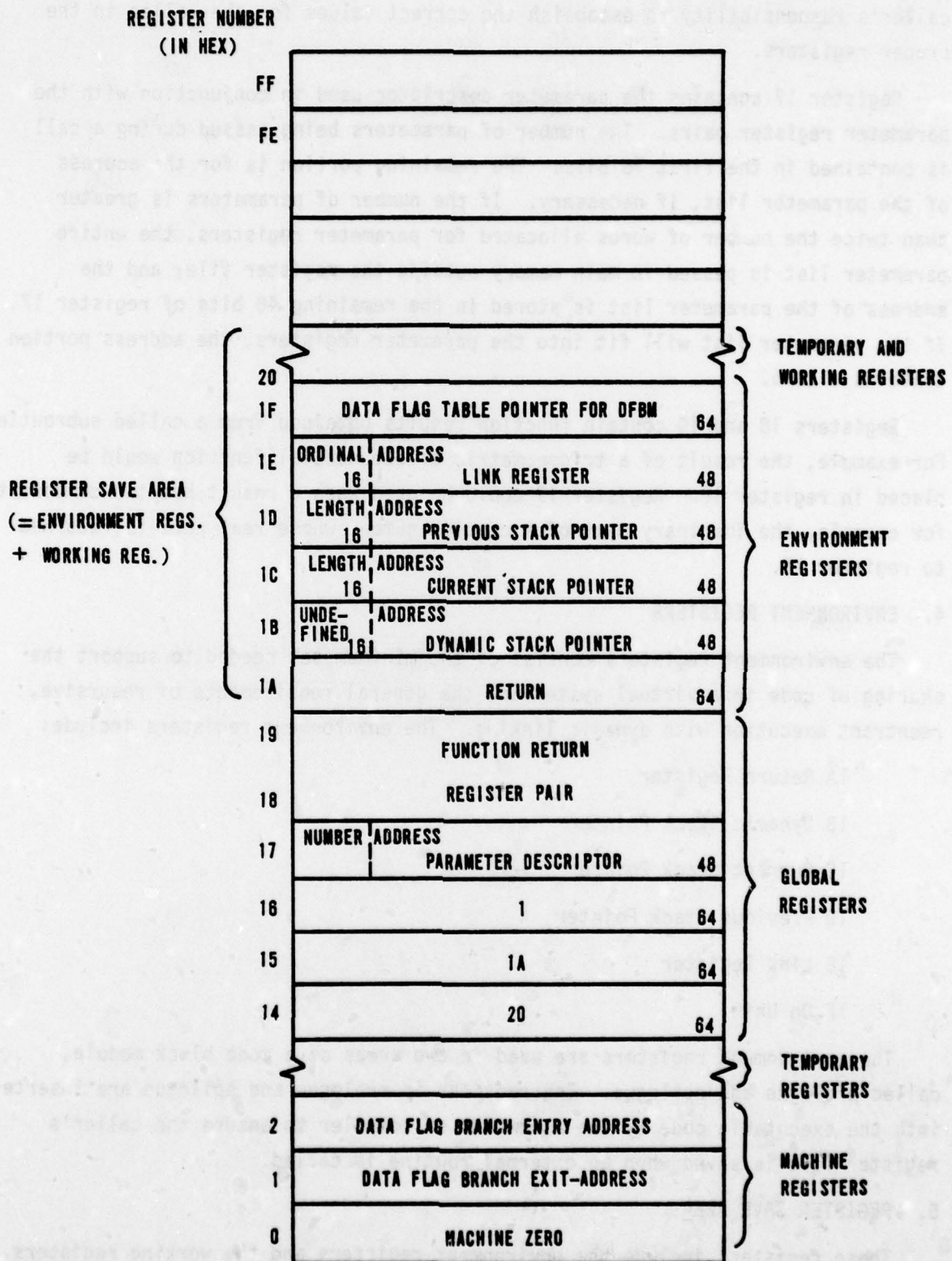


Figure 11. STAR-100 Register File

caller's responsibility to establish the correct values for the callee in the proper registers.

Register 17 contains the parameter descriptor used in conjunction with the parameter register pairs. The number of parameters being passed during a call is contained in the first 16 bits. The remaining portion is for the address of the parameter list, if necessary. If the number of parameters is greater than twice the number of words allocated for parameter registers, the entire parameter list is passed in main memory outside the register file; and the address of the parameter list is stored in the remaining 48 bits of register 17. If the parameter list will fit into the parameter registers, the address portion contains a zero.

Registers 18 and 19 contain function results obtained from a called subroutine. For example, the result of a trigonometric or exponential function would be placed in register 18. Register 19 could be used when a result has two components; for example, the imaginary part of a complex number whose real part is returned to register 18.

#### 4. ENVIRONMENT REGISTERS

The environment registers consist of the minimum set needed to support the sharing of code in a virtual system and the general requirements of recursive, reentrant execution with dynamic linking. The environment registers include:

- 1A Return Register
- 1B Dynamic Stack Pointer
- 1C Current Stack Pointer
- 1D Previous Stack Pointer
- 1E Link Register
- 1F On Unit

The environment registers are used in two areas of a code block module, called prologue and epilogue. Instructions in prologue and epilogue are inserted into the executable code by the assembler or compiler to ensure the caller's register file is saved when an external routine is called.

#### 5. REGISTER SAVE AREA

These registers include the environment registers and the working registers. This space is saved and restored by called processors; therefore it is the space

where permanent variables and addresses should be stored. The length of this area depends on how much space has been allocated by the caller as working registers, which will contain other information the user deems necessary to save. Allocation of environment registers at the beginning of the space ensures that they will appear at the beginning of every stack, facilitating unstacking or stack searching procedures needed for block structured languages, as well as nonstandard FORTRAN call/return usage. The working registers follow the environment registers.

The information in the register save area is used in the following manner: When a program in process calls an external program, the prologue of the called program executes code to save the caller's register file in dynamic space. It then places the current stack pointer in the previous stack pointer, and also places the dynamic stack pointer in the current stack pointer, and sets the dynamic stack pointer to the next free location. Finally, the prologue loads the called program's register file from static space.

When one program calls another, it uses some dynamic space to contain the status of the register file at the time the next program was called, together with linking information required to return to the calling program. In the normal sequence, dynamic space use increases until the lowest level called program has been executed; then, as the returns are encountered, the space is made available in reverse order to the call.

## 6. PARAMETER REGISTER PAIRS

These registers permit a varying number of parameters to be passed via the register file (depending on the execution environment). The parameters are assigned from register FF backward toward the beginning of the register file area. (Thus, if there are five register pairs, the Parameter Register Area will begin at F6.)

All parameters are either passed in the parameter section of the register file, or they are in main memory outside the register file area and are noted in the Parameter Descriptor register (register 17). This register convention allows paired parameter passing of the following form:

- Passing base addresses and corresponding offsets

- Passing pointer pairs for sparse vectors

- Passing double or complex parameters

## APPENDIX J

## STAR-100/INSTRUCTION TYPES

## 1. REGISTER INSTRUCTIONS

The STAR register file consists of 32- and 64-bit registers. To accommodate the use of both register types, the STAR instruction set includes instructions which access the register file as half-words (32 bits) or full-words (64 bits).

In the register instructions, all source and result destinations are registers. Unless specified, in register-to-register operations the source registers are unchanged and the destination registers are cleared before the result is entered.

## 2. INDEX INSTRUCTIONS

Index instructions are used primarily for numerical calculations on field lengths and addresses. The index instructions manipulate either the low order 24 bits of half-word or the low order 48 bits of a full word in designated operational registers. Some index instructions are used for manipulating the high order 8 bits of a half-word or the high-order 16 bits of a full-word in the designated operational registers.

## 3. BRANCH INSTRUCTIONS

The branch instructions can be used to compare or examine single bits, 48-bit indexes, 32-bit floating-point operands, or 64-bit operands. Results of comparison determine whether the program continues with the next sequential instruction (branch condition not met) or branches to a different instruction sequence (branch condition met). The instruction sequence can consist of one or more instructions beginning at the branch address specified in the branch instruction format. For instructions which require index operations, all item counts are in half-word increments.

## 4. VECTOR INSTRUCTIONS

The vector instructions perform operations on ordered elements (scalars). These instructions read the scalars, in 32-bit or 64-bit floating-point operand form, from consecutive storage locations over a specified address range (field).

Vector instructions perform a designated operation on each set of operands and store the results in consecutive addresses of a result field, beginning with a specified address. A vector can contain as many as 65,536 items.

#### 5. SPARSE VECTOR INSTRUCTIONS

Arithmetic operations can reduce the number elements of a vector field to a zero or near-zero value; therefore, except for positional significance, they need not be carried along as floating-point numbers. To conserve both storage and calculation time, a group of sparse vector instructions which permit the expansion and compression of vectors can be used.

#### 6. VECTOR MACRO INSTRUCTIONS

Vector macro instructions perform operations similar to vector instructions; however, some vector macro instructions do not form result vector fields. For these instructions, the control vector contains neither length nor offset; rather it controls the use of source vector elements. The control vector for macro instructions which produce result vector fields, performs the same function as vector instruction. Vector macro instructions with result field(s) extend short source fields with zeros; they become no-operations, and terminate in an identical manner as a vector instruction. Vector macros with result field(s) terminate when either source vector is exhausted; they do not zero extend short source fields.

#### 7. STRING INSTRUCTIONS

String instructions perform arithmetic and logic operations on strings of data in the form of 8-bit bytes. The byte size allows for handling large alphabets (256 characters) and is compatible with ASCII extended binary code. The field length of a data string can be extended beyond one 64-bit word or can be less than one data word. Bytes in the field of a data string are in opposite order of the byte address; the most significant byte is the left-most byte, but the address of the left-most byte is 0. Unless specified by the instruction, strings are processed from right to left until the last byte in the field is processed. Normally, string instructions terminate when the result field is filled.

#### 8. LOGICAL STRING INSTRUCTIONS

These instructions function in the same general manner as corresponding string instructions. They operate with index and data fields the same as string instructions except item counts are expressed in bits instead of bytes; therefore, these instructions perform bit operations on bit boundaries.

## 9. MONITOR INSTRUCTIONS

The monitor instructions function only during monitor mode. When a machine is in job mode, any attempt to execute a monitor instruction is detected by the hardware as an attempt to perform an undefined function code.

## 10. NONTYPICAL INSTRUCTIONS

These instructions perform operations such as register to storage transfers, formation of repeated mask lists, and maximum/minimum determinations that do not belong in any of the preceding instruction types discussed.

## APPENDIX K

## SUMMARY OF MACHINE CHARACTERISTICS

<u>Characteristic</u>	<u>ASC</u>	<u>CRAY-1</u>	<u>STAR-100</u>
<u>Central Processor</u>			
Type	Up to four parallel pipelined units.	Twelve pipelined current functional units.	Two parallel pipelines.
Cycle time	80 nsec	12.5 nsec	40 nsec
<u>Central Memory</u>			
Type	Bi-polar semi-conductor.	Bi-polar semi-conductor.	Core
Maximum size	16,777,216 words.	1,048,576 words.	1,048,576 words.
Maximum Interleaving	8-way	16-way	32-way
Word Size	32-bit	64-bit	64-bit
Transfer Rate	1 word/17.5 nsec	1 word/12.5 nsec	1 word/20 nsec
<u>Instructions</u>			
Size	32-bit	16- and 32-bit	32- and 64-bit
Stack-size	2, 8-word buffers	4, 64-word buffers	4, 8-word buffers

<u>Characteristic</u>	<u>ASC</u>	<u>CRAY-1</u>	<u>STAR-100</u>
-----------------------	------------	---------------	-----------------

Scalar Arithmetic Execution Time

Time in nsec to perform a 64-bit floating point operation on register resident operands to produce a register resident result,

Add	400	75	440
Multiply	480	87.5	600
Divide	2080	362.5	1720

Vector Arithmetic Set-up/Result Time

Time in nsec to set-up a 64-bit vector array for floating point operations/time in nsec to produce a vector result.

Add	$2480 + 560 (N/4)/140$	$200 + 12.5 (N)/12.5$	$2840 + 160 (N/8)/20$
Multiply	$2560 + 240 (N)/240$	$212.5 + 12.5 (N)/12.5$	$6360 + 320 (N/8)/40$
Divide	$4160 + 1520 (N)/1520$	$487.5 + 12.5 (N)/12.5$	$6680 + 640 (N/8)/80$